UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Entrenamiento de un mini-robot para realizar tareas mediante aprendizaje automático

Daniel Cabañas González

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Entrenamiento de un mini-robot para realizar tareas mediante aprendizaje automático

Autor: Daniel Cabañas González
Tutor: Juan Jesús Roldán Gómez

mayo 2021

Daniel Cabañas González
Entrenamiento de un mini-robot para realizar
tareas mediante aprendizaje automático

Daniel Cabañas González
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres y a mi perro

Nadie ha logrado nada de valor sin esforzarse antes por ello.
Es el esfuerzo lo que le otorga valor a las cosas.

Fernando Cabañas

Agradecimientos

En primer lugar, me gustaría darle las gracias a mi tutor por la atención y la ayuda que me ha ofrecido durante la realización de este proyecto.

También quisiera agradecer a toda mi familia, y en especial a mi hermana, por los ánimos y el apoyo constante a lo largo de la carrera y de mi trayectoria académica en general.

Quisiera mencionar también a Gonzalo, a Ignacio, a Pablo y a Guillermo, mis mejores amigos a los que considero mis hermanos.

Por último, me gustaría mencionar a todos los amigos que me han acompañado a lo largo del grado, entre ellos: Santiago, David, Sergio, Alfredo y Álvaro. A todos, muchas gracias por haber estado ahí para mí.

Resumen

El aprendizaje automático y las distintas áreas que lo comprenden están adquiriendo una gran relevancia en diversos campos profesionales y de investigación. Entre estas áreas se encuentra el aprendizaje por refuerzo, que tiene como objetivo el entrenar agentes para que aprendan y desarrollen comportamientos que les permita ejecutar de forma efectiva las tareas propuestas, utilizando sistemas de recompensas que premien o castiguen sus acciones y así condicionar sus comportamientos hasta perfeccionar sus resultados.

Una de las aplicaciones más conocidas en el aprendizaje por refuerzo es el entrenamiento de robots para la realización de tareas sencillas, puesto que existe una fácil implementación de este área a los sistemas robóticos y porque los resultados de aprendizaje obtenidos son satisfactorios.

En este proyecto se ha propuesto simular un modelo de robot real en el entorno de desarrollo Unity, y de entrenar esta simulación aplicando aprendizaje por refuerzo para que consiga resolver misiones de tipo exploración. A través de Unity se incluirá la librería ML-Agents, que permite el desarrollo con herramientas de aprendizaje automático de una forma muy versátil y cómoda.

Se han trabajado con redes neuronales, algoritmos de aprendizaje por refuerzo, sistemas de recompensas y gestión de entradas y salidas entre otras decisiones que han permitido desarrollar modelos capaces de solucionar entornos laberínticos.

Los modelos de aprendizaje automático han entrenado en entornos simples y complejos, llegando a resolver laberintos de dimensiones 3x3 en aproximadamente 4 minutos durante los entrenamientos y siendo capaces de resolver laberintos de dimensiones 5x5 mediante generalización en aproximadamente 15 minutos sin haber entrenado en ellos previamente.

Palabras clave

Aprendizaje Automático, Aprendizaje por Refuerzo, Robótica, Robot, Simulación, Misiones de Exploración

Abstract

Machine learning and the different areas within it are gaining great relevance in multiple professional and research fields. Among these areas we can find reinforcement learning, which aims to train agents to learn and develop behaviors that allow them to effectively perform the proposed tasks, using reward systems that compensate or punish their actions and condition their behaviors to improve their results.

One of the best known applications in reinforcement learning is the training of robots to carry out simple tasks, since there is an easy implementation of this area to robotic systems and the learning results obtained are suitable.

In this project we have proposed to simulate a real robot model in the Unity development environment, and train this simulation by applying reinforcement learning so that it can solve exploration-type missions. Through Unity, the ML-Agents Package will be included, which allows working with machine learning tools in a very adaptable and comfortable way.

We have worked with neural networks, reinforcement learning algorithms, reward systems and management of inputs and outputs among other decisions that have allowed the development of models capable of solving labyrinthine environments.

Our machine learning models have trained in simple and complex environments, sorting out 3x3 dimensional mazes in approximately 4 minutes during training and being able to solve 5x5 dimensional mazes through generalization in approximately 15 minutes without having previously trained in them.

Keywords

Machine Learning, Reinforcement Learning, Robotics, Robot, Simulation, Exploration Missions

ÍNDICE

Listas

Lista de algoritmos

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

En la actualidad existen una gran cantidad de tareas llevadas a cabo por robots o por máquinas debido a su complejidad, a que estas tareas tienen lugar en entornos peligrosos o simplemente por la eficiencia lograda respecto a la que pueden alcanzar los seres humanos. A día de hoy, el número de tareas de este tipo va en aumento, y a su vez aparecen nuevos paradigmas que limitan las posibilidades de los robots, tanto en el diseño físico como en su propia programación.

Por ello, durante estos últimos años se ha estado investigando y avanzando a pasos agigantados en los campos de la inteligencia artificial y el aprendizaje automático aplicados a la robótica y la mecatrónica, ya que gracias a ellos se consigue abordar problemas que no se pueden resolver con la programación tradicional.

El objetivo de este TFG es realizar un ensayo sobre la aplicación del aprendizaje automático, más en concreto el aprendizaje por refuerzo, en un pequeño robot de LEGO® MINDSTORMS®. Este robot deberá resolver problemas de exploración de áreas y de navegación en laberintos desconocidos. La resolución de este tipo de problemas tiene aplicaciones en cantidad de ámbitos tecnológicos que plantean dificultades en la toma de decisiones, y que las técnicas tradicionales de algoritmia no son capaces de resolver de forma eficiente.

Este TFG pertenece a una línea de investigación junto con otro TFG paralelo, que tiene como finalidad integrar el robot real con un ordenador con la intención de poder implementar algoritmos complejos con él. Se plantea la unificación de estos dos proyectos en un futuro para poder implementar modelos entrenados con aprendizaje por refuerzo en el propio robot real, y comprobar que sea capaz de resolver misiones de exploración a través de su aprendizaje inducido.

El proyecto se centrará en el modelado del robot en una plataforma virtual donde se realizarán entrenamientos utilizando redes neuronales conocidas por su aplicación en el campo del aprendizaje por refuerzo. Para ello, se deben analizar el robot y sus componentes (sensores y actuadores), abstrayendo sus características como: sus posibles velocidades lineales y velocidades angulares, su peso o sus dimensiones.

Una vez conseguido el modelo virtual del robot, se procederá al entrenamiento de este en entor-

nos laberínticos complejos, con distintas características y estructuras, para alcanzar un aprendizaje más completo. Así mismo, se implementarán varios entornos simultáneos para lograr un aprendizaje paralelo que será más rápido y eficiente.

Por último, se contrastarán los resultados obtenidos de los entrenamientos del modelo virtual del robot, siguiendo un proceso en el que la dificultad de éstos irá en aumento, y a su vez comparando distintas configuraciones para poder destacar cuáles obtienen los mejores resultados.

1.1. Motivación

El área de la robótica ha evolucionado en gran medida durante los últimos años, logrando producir máquinas cada vez más complejas tanto en el diseño físico como en su propia programación.

El campo de la inteligencia artificial, perteneciente a la rama de la cibernética, es uno de los campos que más está creciendo en el área de la robótica actualmente, ya que su objetivo es conseguir desarrollar máquinas inteligentes que sean capaces razonar y decidir como hacen los seres humanos. Con ello se pretende encontrar una solución a problemas que los robots tradicionales cuyos comportamientos limitados y poco flexibles no pueden hallar de una forma eficiente.

El objetivo principal de este área es crear robots “inteligentes” que posean una autonomía que les permita resolver los problemas para los que fueron diseñados. Esta autonomía está basada en la capacidad del robot de analizar su entorno, entenderlo y adaptarse a él, tomando las mejores decisiones para desarrollar su labor.

Dentro de la inteligencia artificial se han desarrollado recientemente distintas técnicas de aprendizaje automático. En este proyecto se trabajará con aprendizaje por refuerzo, uno de los tipos de aprendizaje automático más prometedores en la robótica junto con el aprendizaje por imitación.

El aprendizaje por refuerzo permite condicionar el comportamiento de la máquina o robot para que este tome las decisiones correctas a la hora de resolver los problemas que se le han planteado, logrando objetivos de forma eficiente que con un paradigma tradicional no podría haber conseguido.

Se prevé que estos avances en la robótica tendrán un impacto muy relevante en las áreas de la mecánica, la electrónica y la computación, y que sus aplicaciones irán desde la industria tecnológica hasta los ámbitos sociales como pueden ser la salud y la seguridad.

1.2. Objetivos

En este trabajo se proponen técnicas de aprendizaje por refuerzo para que un robot móvil de pequeño tamaño sea capaz de navegar por un laberinto de estructura desconocida. Para lograr este

trabajo se consideran los siguientes objetivos:

1.– Estudiar el funcionamiento del robot con sus sensores y sus motores.

2.– Implementar el comportamiento del robot en la plataforma Unity, incluyendo sus dimensiones, peso y configuración de movimientos.

3.– Implementar las funcionalidades de los sensores del robot real al modelo simulado en Unity.

4.– Reproducir uno o varios entornos virtuales en la plataforma Unity para que el robot realice su misión.

5.– Demostrar que el modelo del robot virtual coincide con el modelo real para los restantes objetivos propuestos.

6.– Demostrar que se produce un aprendizaje con las técnicas propuestas. Exponer las mejoras en los entrenamientos con el fin de destacar un comportamiento que consiga progresar.

7.– Demostrar que el modelo es capaz de acometer la misión de exploración en un laberinto de estructura desconocida.

1.3. Organización de la Memoria

La memoria ha sido dividida en los siguientes apartados. El primero de ellos, Estado del Arte, será utilizado para introducir el contexto teórico relacionado con el aprendizaje automático y el aprendizaje por refuerzo aplicado a la robótica. El segundo apartado, Diseño, contiene una descripción de los elementos y herramientas utilizadas a lo largo del Trabajo de Fin de Grado. En el apartado de Desarrollo se explicará la trayectoria que ha seguido el proyecto desde los primeros pasos realizando mediciones del robot real hasta el diseño de las pruebas finales. En el apartado de Integración, Pruebas y Resultados se llevarán a cabo una explicación y descripción de las pruebas realizadas a lo largo del proyecto, y se comentarán los resultados obtenidos en ellas. La memoria finaliza con el apartado de Conclusiones y Trabajo Futuro, donde se comentarán los objetivos cumplidos en el proyecto y las posibilidades que este ofrece de cara al futuro.

2

Estado del Arte

En la actualidad, la inteligencia artificial es utilizada en muchos y diversos aspectos de nuestra vida, desde la detección y prevención de enfermedades en el campo de la medicina [1] hasta la gestión y análisis de entrevistas de trabajo en departamentos de *marketing* [2].

No obstante, antes de la aparición de la inteligencia artificial la programación estaba supeditada a que el programador diseñase previamente el comportamiento lógico de los sistemas de computación para que estos funcionasen correctamente.

La programación tradicional consiste en diseñar de manera manual una serie de instrucciones y normas en un orden específico con la intención de elaborar un proyecto o sistema que produzca unos resultados establecidos con anterioridad. El objetivo de este tipo de planteamientos es encontrar una solución eficiente a un problema concreto, mediante el uso de algoritmos, operaciones, variables y otros elementos destacables del entorno de la programación. Sin embargo, en ciertas situaciones específicas este tipo de programas pueden no ser capaces de encontrar soluciones eficientes o simplemente no conseguir resolver el problema para el que son diseñados.

Con la aparición de la inteligencia artificial se plantea un nuevo paradigma donde se parte del trabajo con resultados para producir estructuras y procedimientos que puedan alcanzar más resultados del mismo tipo. De esta manera, una de las ventajas del campo de la inteligencia artificial es que la cuestión “qué debemos alcanzar” puede ser sustituida por el “cómo podemos alcanzarlo”.

La inteligencia artificial también se diferencia de la programación clásica en otros aspectos. Por un lado, existen situaciones en las que es difícil conocer toda la información necesaria para resolver los problemas con la metodología tradicional, o donde esta metodología no logra resultados eficientes, como es el ejemplo de los algoritmos genéticos aplicados a los problemas NP-complejos, capaces de encontrar soluciones mucho más eficientes que cualquier planteamiento clásico [3].

Por otro lado, uno de los aspectos más relevantes de la inteligencia artificial la encontramos en el concepto de ‘clasificación’, cuyo objetivo es ordenar los datos según una serie de criterios, permitiendo así destacar atributos significativos para obtener resultados concretos. Existen múltiples clasificadores y algoritmos que se diferencian en su modo de tratar la información, y que logran mejores o peores

resultados en función del tipo de problema en el cual estén siendo aplicados.

Dentro de las múltiples ramas de la inteligencia artificial, este Trabajo de Fin de Grado pone el foco en la rama del aprendizaje automático, la cual está centrada en el desarrollo de programas y modelos capaces de mejorar sus propios comportamientos, logrando resultados cada vez más precisos y eficientes. Después, se comentarán los tipos de aprendizaje más relevantes y con ello se dará paso al aprendizaje por refuerzo, que es el tipo de aprendizaje automático que se aplica en este proyecto.

2.1. Aprendizaje Automático

El *machine learning* o aprendizaje automático es una rama del campo de la inteligencia artificial que está enfocada en lograr un aprendizaje por parte de un programa o un modelo.

Se entiende por aprender el hecho de que un programa o modelo sea capaz de recolectar información (de su entorno a través de sensores, o aportada a través de un banco de datos), analizar y estructurar esta información y realizar una serie de conclusiones con las que más adelante poder tomar decisiones para realizar las tareas que logren su objetivo, todo ello con un carácter progresivo que consiga cada vez mejores resultados.

Una definición esquemática muy apropiada es la que hace Moreno en su libro [4], donde comenta que el aprendizaje de una máquina o programa se podría resumir en una selección de ciertas características o atributos dentro de la información recogida previamente por parte del programa, para más adelante adaptar su modelo o comportamiento en función de la información destacada.

Una vez definido el concepto de aprendizaje, se puede comprender que el aprendizaje automático consta de tres partes diferenciadas:

- Información: Son los datos almacenados inicialmente o recabados por la máquina o programa. Esta información puede ser definitiva o ir variando en función del problema planteado (ej. si es una tarea de exploración, la máquina o programa irá renovando la información a través de sus sensores).
- Modelo: Es el algoritmo o esquema que utiliza el programa o máquina para proceder. Define el comportamiento de esta basándose en la información recabada y seleccionada con anterioridad junto con nueva información, produciendo unos resultados o decisiones que determinarán las acciones a realizar.
- Resultados: Son los datos de salida producidos por el modelo. Puede ser sencillamente información (p. ej. utilizada por analistas) o instrucciones que la máquina o programa usará para realizar ciertas acciones (como determinar el próximo movimiento de un robot). En el aprendizaje automático el modelo suele utilizar sus propios resultados, añadiéndolos a la base de información para utilizarlos en iteraciones futuras.

Se puede observar un resumen del comportamiento de un programa o máquina que utiliza aprendizaje automático en la figura 2.1.

Como se ha comentado anteriormente, una de las características principales del aprendizaje automático, y una de sus principales ventajas frente a la programación tradicional, es su flexibilidad y

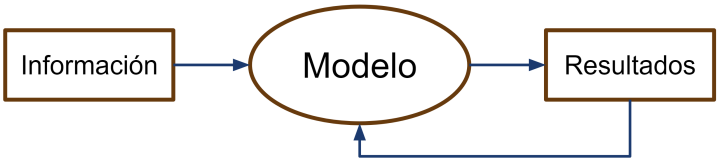


Figura 2.1: Esquema de comportamiento del aprendizaje automático

capacidad de adaptación a la hora de trabajar con nueva información. Este es uno de los motivos por los que esta rama tiene tanto éxito en el ámbito de investigación y en ámbito profesional, puesto que permite fácilmente trabajar con bancos de datos, que están renovando su información constantemente, sin tener que reprogramar o reconfigurar los algoritmos utilizados. Un ejemplo dentro de las múltiples aplicaciones del aprendizaje automático en el campo de la medicina es el de crear algoritmos cuyos diagnósticos compiten con el de los médicos humanos, ya que ambos poseen una base que parte de la estadística [5].

En el aprendizaje automático se emplean distintos tipos de algoritmos y técnicas, cada una de ellas apropiada para diferentes casos, y que se verán en la siguiente sección.

2.2. Tipos de Aprendizaje Automático

A lo largo de la historia del aprendizaje automático se han desarrollado distintas técnicas o algoritmos debido a la aparición de nuevos paradigmas como el procesamiento de imágenes, la minería de datos, etc. Estas nuevas metodologías son más efectivas que sus predecesoras, ya sea porque los anteriores algoritmos no resultaban suficientemente eficientes o simplemente porque se mejoran creando versiones más competentes.

A la hora de catalogar los distintos tipos de algoritmos o procedimientos, existen múltiples clasificaciones por parte de distintos autores y entidades, aunque estas suelen ser muy parecidas. En este proyecto se ha decidido seguir la clasificación de Brownlee [6], donde se distinguen los distintos campos de investigación, las técnicas híbridas entre algunos tipos de aprendizajes y otras variantes a tener en cuenta.

2.2.1. Tipos Principales de Aprendizaje

Los tipos de aprendizaje más famosos y relevantes son el aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo, ya que estos tres tipos de aprendizaje abarcan la mayoría de los problemas que se resuelven mediante la aplicación del aprendizaje automático.

A continuación, se realizará una introducción al aprendizaje supervisado y no supervisado única-

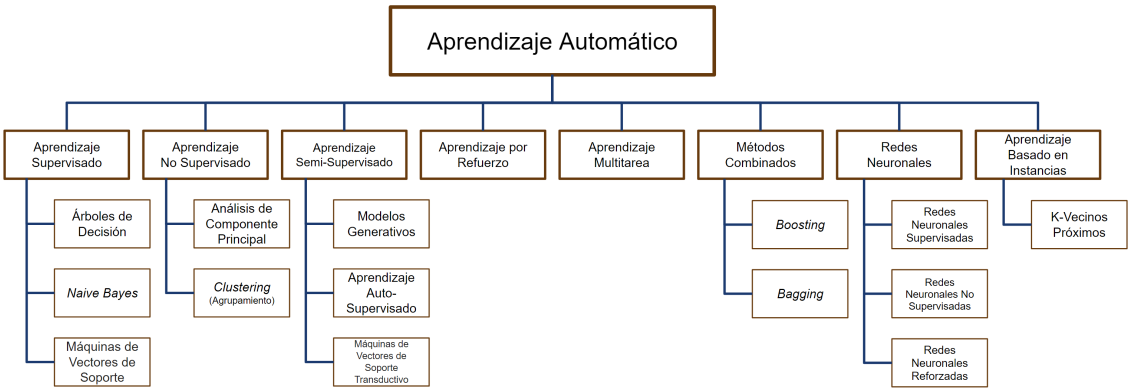


Figura 2.2: Ejemplo de clasificación de tipos de aprendizaje automático, adaptado de [7]

mente, ya que más adelante se pondrá el foco en el aprendizaje por refuerzo, el cual aplicamos en este proyecto.

Aprendizaje Supervisado

El aprendizaje supervisado consiste en desarrollar algoritmos que utilizan un conjunto de datos denominados instancias (las cuales constan de entradas y salidas esperadas) para entrenar un modelo. El modelo deberá desarrollar una relación entre las entradas (comúnmente denominadas atributos) y la salida (pudiendo ser esta una clase o un valor). Más adelante, este modelo deberá intentar predecir las salidas de instancias futuras etiquetándolas entre las posibles clases o valores, ayudándose de la configuración adquirida durante el entrenamiento [8].

Entre los principales problemas de aprendizaje supervisado se encuentran el aprendizaje supervisado por clasificación, que consiste en predecir una etiqueta o categoría [9], y el aprendizaje supervisado por regresión, que consiste en predecir un valor numérico [10].

Aprendizaje No Supervisado

El aprendizaje no supervisado difiere del aprendizaje supervisado en que las instancias de datos que se le entrega para su entrenamiento únicamente contienen entradas y no salidas. No existe una referencia en cuanto al etiquetado o enmarcación de las distintas clases posibles, puesto que el objetivo del aprendizaje no supervisado es encontrar relaciones entre los atributos de las distintas instancias.

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado no tiene una respuesta acertada como tal, y se le designa como “no supervisado” porque no existe un control o corrección en las respuestas que entrega [12].

Los tipos de aprendizaje no supervisado más destacables son el denominado *clustering* (o agrupamiento en español) y las reglas de asociación:

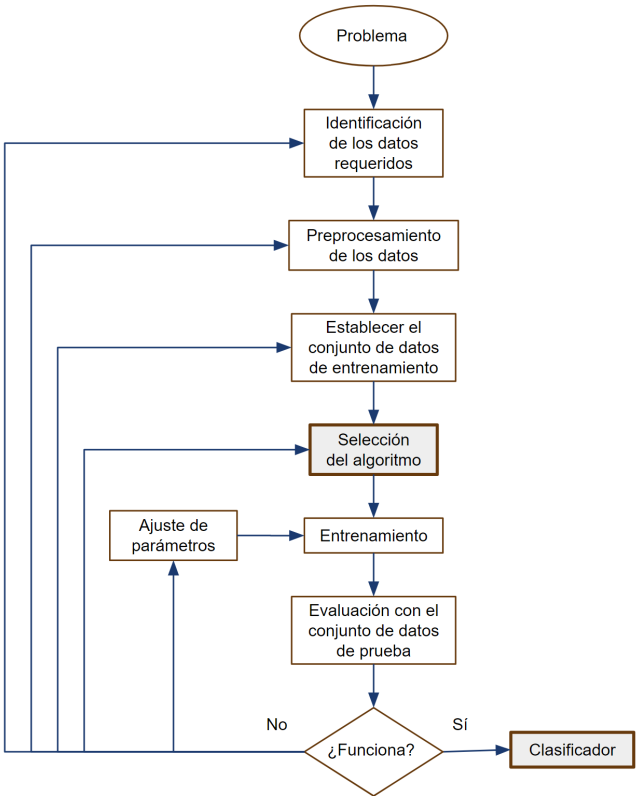


Figura 2.3: Esquema del algoritmo de aprendizaje supervisado, adaptado de [11]

El *clustering* es un tipo de problema que consiste en agrupar porciones de instancias en función de ciertas características [13], como podría ser una serie de clientes de un banco que tienen una edad aproximada.

Las reglas de asociación se utilizan para destacar ciertas normas o tendencias que aparecen en los conjuntos de datos [14]. Un ejemplo sería destacar una tendencia que muestre que un tipo de cliente, que haya comprado un producto específico, sea propenso a comprar más adelante otro producto distinto. Este tipo de información es muy interesante para que los negocios y las empresas puedan predecir los intereses de sus clientes.

2.2.2. Tipos de Aprendizaje Híbridos

En esta sección se comentará los tipos de aprendizaje apodados como “híbridos”. Esto es así, debido a que estos tipos de aprendizaje no forman parte únicamente del supervisado o no supervisado, sino que se encuentran entre ambos campos de estudio.

Entre los varios tipos de aprendizaje híbridos, se pueden destacar tres de ellos:

- Aprendizaje Semi-Supervisado: Consiste principalmente en un tipo de aprendizaje supervisado, donde el conjunto de datos de entrenamiento está compuesto por una gran cantidad de datos sin etiquetar (es decir, sin una salida indicada) y una pequeña porción de datos etiquetados. Esto se hace para que el modelo que se está

entrenando tenga que encontrar mayores relaciones (y más complejas) entre los atributos para poder etiquetarlos, evitando así que el modelo únicamente se dedique a etiquetar. El aprendizaje semi-supervisado se utiliza fundamentalmente para trabajar con conjuntos de datos de carácter complejo, que contengan instancias cuyas relaciones entre ellas suelen ser débiles [15].

- Aprendizaje Multi-Instancia: En el aprendizaje multi-instancia cada muestra de aprendizaje es representada con una “bolsa” de instancias en vez de las propias instancias de manera individual. Las instancias no están etiquetadas de por sí, mientras que la bolsa sí está asociada a una etiqueta. Se utiliza el término “bolsa” y no conjunto debido a que pueden existir instancias duplicadas dentro de una misma bolsa [16] [17].
- Aprendizaje Auto-Supervisado: El aprendizaje auto-supervisado es aquel que trata un problema de aprendizaje no supervisado con un marco de aprendizaje supervisado. Es decir, existe una supervisión automática (sin intervención humana) y una categorización por parte del algoritmo, el cual trabaja con un conjunto de datos que no están inicialmente etiquetados, y es el mismo algoritmo el que trata de etiquetarlos en función de las asociaciones que destaca entre los atributos de las entradas, todo ello en un ciclo de retroalimentación [18].

2.3. Aprendizaje por Refuerzo

El aprendizaje por refuerzo no contiene un conjunto de datos etiquetados, por lo que no se le puede enmarcar en el aprendizaje supervisado, y como recibe ciertas indicaciones sobre su progreso tampoco se le puede enmarcar en el aprendizaje no supervisado. Por ello, el aprendizaje por refuerzo es considerado el tercer tipo principal de aprendizaje dentro del aprendizaje automático.

El aprendizaje por refuerzo plantea una disposición donde un agente interactúa con un entorno (comúnmente a través de sensores u otro tipo de mecanismos) recogiendo información sobre este. El agente tendrá que realizar una serie de acciones, condicionadas por la información recogida y por una puntuación que se genera mediante un sistema de *feedback* o realimentación numérico automático.

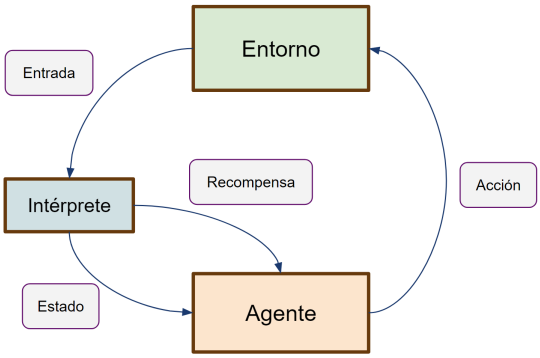


Figura 2.4: Esquema de funcionamiento del aprendizaje por refuerzo, adaptado de [19]

En otras palabras, el agente estará al tanto de su entorno en medida de lo posible, y según qué acción decida tomar en todo momento, se le recompensará o se le penalizará, condicionando así su comportamiento o modelo. El sistema de recompensas tendrá en cuenta dos factores: el estado en el que se halla el agente y las acciones que este realiza.

El objetivo del agente es alcanzar la máxima recompensa posible dentro de una iteración (el evento, experimento o prueba que esté realizando), y para ello debe descubrir por sí mismo la combinación de acciones que le proporcione esta recompensa.

Por ejemplo, en el estudio *Playing Atari with Deep Reinforcement Learning* [20] utilizan una red neuronal convolucional entrenada con aprendizaje por refuerzo para que juegue a distintos juegos de la consola Atari 2600 (donde las entradas son píxeles y cuya salida es una función de estimación de recompensa). En el estudio muestran que esta técnica no solo consiguió mejores resultados que cualquiera de las técnicas utilizadas previamente, sino que además la red neuronal consiguió superar a los seres humanos en algunos de los juegos.

Otro de los ejemplos más célebres en la historia del aprendizaje por refuerzo es el caso del programa “AlphaGo” [21] de Google, la primera inteligencia artificial capaz de ganar a un profesional en una partida del juego Go. Más adelante, se desarrollaron nuevas versiones mucho más potentes, como AlphaZero y AlphaGo Zero.

2.3.1. Aprendizaje por Refuerzo Aplicado a la Robótica

El aprendizaje por refuerzo es frecuentemente utilizado en el área de la robótica debido a que es capaz de solucionar los problemas planteados de forma eficiente, especialmente en aquellos problemas cuyas soluciones son consideradas irregulares o desconocidas [22].

A su vez, la implementación del aprendizaje por refuerzo es muy directa en el ámbito de la robótica, donde es sencillo entender las relaciones entre los elementos que componen el aprendizaje por refuerzo y las partes de un robot. El propio robot suele tener una serie de sensores (estos pueden ser sensores ultrasonidos, cámaras, acelerómetros, giroscopios, sistemas GNSS...) con los que captar información acerca del entorno, la cual será transformada en entradas para el modelo entrenado. El modelo producirá una serie de salidas que el robot traducirá como acciones a través de actuadores (como motores y servos).

La utilidad del aprendizaje por refuerzo es aplicada en múltiples problemas del ámbito robótico, desde desafíos sencillos como conseguir que un robot evite obstáculos en un recorrido hasta problemas más complejos como afinar la eficacia de los motores de un robot que realiza tareas de precisión [23].

En este proyecto se aplica el aprendizaje por refuerzo para entrenar una red neuronal con uno de los algoritmos más punteros en el campo, denominado PPO (*Proximal Policy Optimization*, Optimización de Política Proximal en castellano), del cual se hablará con mayor detalle en el capítulo 3. El modelo recibirá una serie de entradas provenientes de los sensores del robot, y con ellas producirá salidas que se traducirán en instrucciones para que el robot se mueva, pudiendo así resolver distintos circuitos.

2.3.2. Aprendizaje por Refuerzo Frente a Aprendizaje por Imitación

El aprendizaje por imitación es otra rama del aprendizaje automático que también destaca en el ámbito de la robótica debido a que logra resultados óptimos con un entrenamiento eficiente y que, junto al aprendizaje por refuerzo, abarca una gran parte del desarrollo con robots en el campo de la inteligencia artificial.

El objetivo del aprendizaje por imitación es el de conseguir que una máquina o programa sea capaz de imitar el comportamiento de un usuario. El desarrollador facilita ejemplos de comportamiento al agente, el cual intentará mimetizar estos ejemplos (con un grado de adaptabilidad, puesto que el objetivo no es imitar a la perfección, sino aprender de los movimientos de los ejemplos y del por qué de estos), y en función de la semejanza de los resultados y de la eficacia de estos recibirá una realimentación con la que poder seguir ajustando su entrenamiento [24].

Otro de los objetivos propuestos en los modelos de aprendizaje por imitación es el de alcanzar la generalización, logrando superar el comportamiento del usuario aplicando conductas propias desarrolladas por el modelo.

Gracias a lo comentado anteriormente, se puede destacar que el aprendizaje por imitación es un planteamiento que parte del aprendizaje supervisado, ya que contiene un conjunto de datos de referencia con los que poder calibrar sus acciones, y del aprendizaje por refuerzo, puesto que se tiene en cuenta los resultados no solo en la similitud sino en la eficacia a la hora de resolver el problema planteado [25].

En el aprendizaje por imitación se desarrolla una fuerte dependencia de los ejemplos provistos, y por ello se logran resultados muy eficientes en tareas específicas. Pero el aprendizaje por imitación no está muy cualificado para realizar tareas cuyas soluciones sean variables o desconocidas.

Frente a este tipo de problemas y de cara a la posibilidad de investigar métodos alternativos de resolución, el aprendizaje por refuerzo es mucho más competente. Uno de los objetivos de este proyecto es poner a prueba algoritmos de aprendizaje por refuerzo para observar la capacidad que tienen de ampliar su conocimiento adquirido en los entrenamientos a la hora de resolver laberintos de estructura desconocida, un tipo de problema que no se puede resolver correctamente con aprendizaje por imitación.

3

Diseño

El objetivo del proyecto es el de representar el comportamiento del robot de LEGO® MINDSTORMS® en un entorno virtual para poder simular entrenamientos de aprendizaje automático con este modelo virtual. Se ha decidido que la plataforma de desarrollo Unity era la apropiada debido al motor de físicas que posee y a la cantidad de facilidades que otorga a la hora de implementar este tipo de proyectos. Además, a través de Unity se utilizará la librería de aprendizaje automático ML-Agents, que permite utilizar una gran cantidad de herramientas y algoritmos para el desarrollo de proyectos de *machine learning*.

3.1. Robot de LEGO® MINDSTORMS®

El robot que se utiliza en este proyecto es un robot de LEGO® MINDSTORMS®, una plataforma de *software* y *hardware* producida por la marca de juguetes LEGO® para el desarrollo de robots programables con un propósito educacional.

Este robot es el modelo NXT de 2006, y está constituido por un *kit* de piezas que incluye sensores, actuadores, cables, ruedas, piezas para la estructura del robot y una controladora comúnmente denominada *brick* o ladrillo. Se adjunta las especificaciones del robot en el apéndice A.

En los posteriores apartados se introducirán los distintos componentes del robot que se pretenden simular, divididos entre sensores y actuadores.

3.1.1. Sensores

En la versión NXT del robot es posible acoplar diversos tipos de sensores, desde sensores de luz hasta sensores de sonido. A continuación, se expondrá una lista con los sensores que contiene nuestro robot junto con una explicación de sus utilidades. También se incluye una ilustración del robot que detalla la localización de los sensores.

- Sensor Ultrasónico: El sensor ultrasónico mide distancias mediante el uso de ondas ultrasónicas (emite una onda y la recibe una vez que esta rebota, calculando el tiempo desde que se emite hasta que vuelve). Se encuentra en

la parte frontal superior del robot, orientado hacia el frente y está acoplado a un servomotor que le permite rotar. El objetivo de este acoplamiento es que el sensor pueda alcanzar un rango de medición de más de 90 grados, pudiendo así detectar elementos en su frontal, sus diagonales delanteras y parte de sus laterales.

- Sensor de contacto o fin de carrera: El sensor de contacto es un sensor mecánico que genera señales si es accionado (actúa como un interruptor o un pulsador). El robot lleva dos sensores acoplados en la parte frontal inferior del robot (uno en la parte derecha y otro en la izquierda) y ambos, junto con una extensión con forma de parachoques, forman un sistema de detección de colisiones que abarca toda la parte frontal incluyendo las diagonales delanteras.
- Sensor de color: El sensor de color detecta la intensidad de la luz recibida de los colores rojo, verde y azul, siguiendo el modelo de color RGB, para determinar el color del elemento que se esté midiendo. El robot tiene acoplado el sensor de color en la parte frontal inferior del robot, justo detrás de los sensores de contacto. El sensor está orientado hacia el suelo, con el objetivo de poder detectar el color de la superficie donde se encuentra el robot. Esto será útil a la hora de resolver laberintos, ya que utilizaremos plataformas de distintos colores para designar los objetivos dentro del laberinto. Podemos observar un ejemplo del uso de este tipo de sensores en el estudio [26].
- Sensor giroscópico o giroscopio: El sensor giroscópico mide la velocidad angular del robot y los cambios en la rotación de este. Esta información se utiliza para realizar cálculos sobre el movimiento del robot junto con el acelerómetro, y el sensor se encuentra localizado en la parte superior izquierda del robot.
- Sensor de aceleración o acelerómetro: El sensor de aceleración se utiliza para medir cambios en las velocidades del robot. Se utiliza junto con el giroscopio para determinar la posición y los movimientos del robot, y se encuentra acoplado en la parte superior derecha de este.

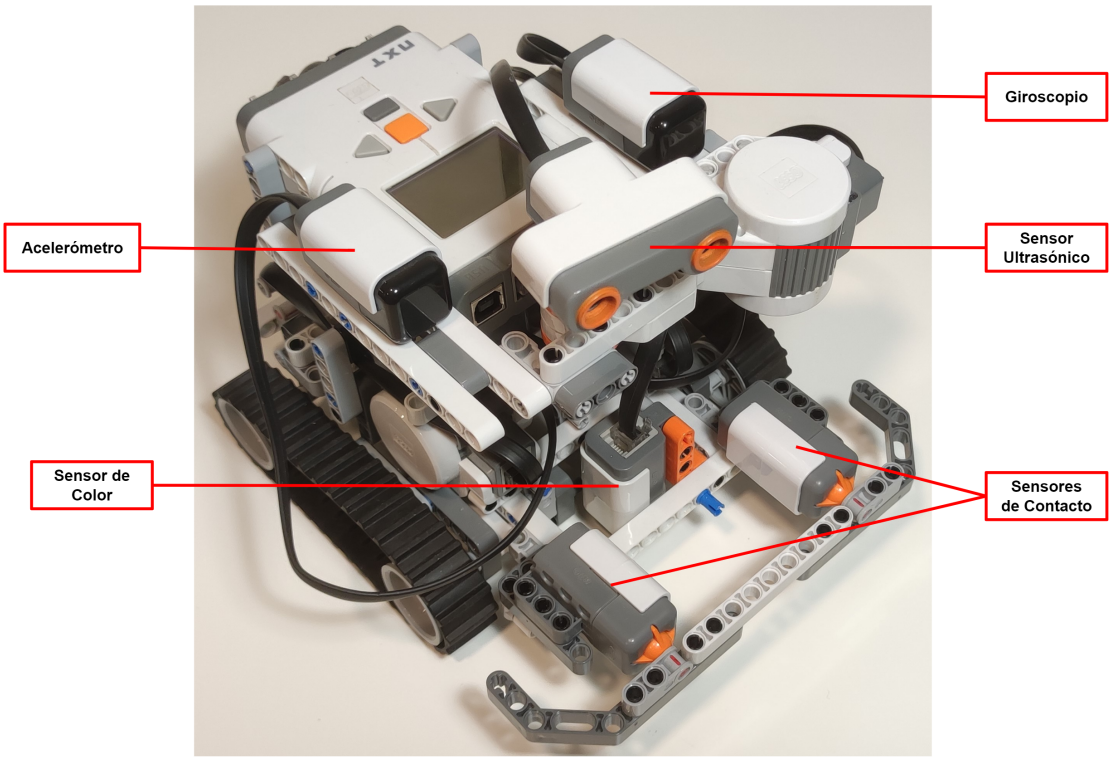


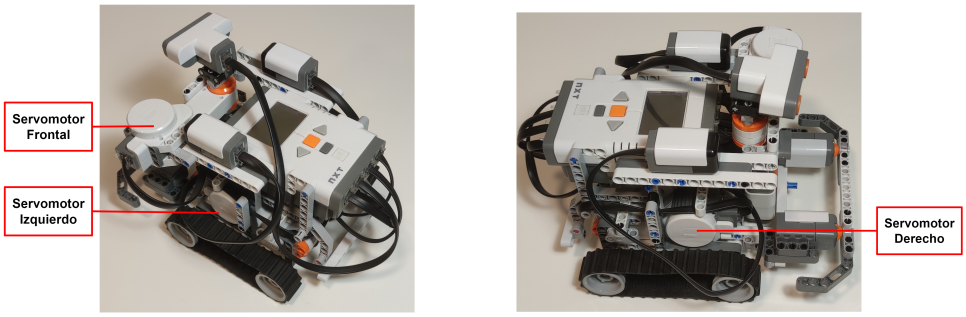
Figura 3.1: Sensores del robot NXT utilizado en este proyecto

3.1.2. Actuadores

Los actuadores son elementos mecánicos cuyo propósito es el de generar una fuerza para realizar una acción o un movimiento. Los actuadores del robot NXT son servomotores eléctricos y son un total de tres.

Los dos primeros servomotores están encargados del desplazamiento del robot, y se encuentran localizados en la parte inferior trasera del robot (uno a cada lado), proporcionando una fuerza de rotación a las ruedas del robot. El robot consta de cuatro ruedas, unidas dos a dos lateralmente por una cinta, formando un sistema de tracción de oruga. Los servomotores actúan rotando las ruedas traseras y mediante la cinta se reparte la tracción a las ruedas delanteras, y ambos son independientes el uno del otro, permitiendo así la rotación en distintas direcciones o velocidades y así dotando al robot de un gran conjunto de movimientos como pueden ser avanzar en línea recta, retroceder, girar o rotar sobre sí mismo.

El tercer servomotor ha sido mencionado anteriormente, ya que es el encargado de rotar el sensor ultrasónico, y se encuentra en la parte frontal superior del robot.



(a) Servomotores frontal e izquierdo (b) Servomotor derecho

Figura 3.2: Actuadores del robot NXT utilizado en este proyecto

3.1.3. Bricx Command Center

LEGO® ha desarrollado una herramienta para la programación del robot conocida como *LEGO® MINDSTORMS® NXT Software*, que distribuye de manera gratuita y que posee un carácter educativo.

Sin embargo, este método de programación se encuentra muy limitado respecto a las opciones que ofrece, y por ello en este proyecto se ha utilizado el entorno de desarrollo *Bricx Command Center* [27], que permite programar distintos comportamientos en el lenguaje *NXC*, el cual es muy parecido al lenguaje de programación *C*. Se han desarrollado distintos comportamientos utilizando BCC con el objetivo de probar las configuraciones de movimiento y de los sensores que se pretenden implementar en los entornos de aprendizaje. Todo ese proceso será detallado en el capítulo 4.

3.2. Unity

Unity es un motor de desarrollo (o motor gráfico) conocido por ser usado principalmente para realizar videojuegos. Un motor de desarrollo es una herramienta de *software* que permite diseñar y producir entornos interactivos.

Entre los muchos elementos incluidos en este motor de físicas, se puede destacar el componente *rigidbody*, un atributo que se acopla a los objetos del entorno y permite la detección de colisiones, establecer velocidades y aplicar distintas fuerzas entre otros. Gracias a este componente los objetos adquieren comportamientos similares a los de la vida real en cuanto a masa o gravedad se refiere.

Otro elemento a destacar es el componente *collider*, una estructura geométrica que permiten simular impactos y choques entre elementos dentro del entorno, ya que *rigidbody* solo se encarga de detectar estas colisiones.

Por último, se comentará la utilidad de las interfaces inmersivas en Unity, las cuales tienen aplicaciones directas en cuanto a la manipulación y control de robots [28], y la interconexión de Unity con ROS (*Robot Operating System*), un conjunto de herramientas y librerías para el diseño e implementación de robots en una amplia variedad de plataformas [29], muy interesante de cara al futuro que podría tomar el proyecto.

Como se ha mencionado antes, Unity reúne las características esenciales para ser elegido como herramienta de desarrollo de videojuegos. Sin embargo, gracias a su comunidad *online*, Unity también incluye una gran cantidad de librerías que añaden diversas funcionalidades como pueden ser las herramientas matemáticas de la librería Unity3D-MatLab [30] o la propia librería de ML-Agents para trabajar con algoritmos de aprendizaje automático, de la cual se hablará a continuación ya que forma uno de los pilares más importantes en el desarrollo de este Trabajo de Fin de Grado.

Entre las muchas versiones que existen de Unity, en este proyecto se ha utilizado la versión 2019.3 [31] y se ha trabajado con una licencia proporcionada de forma gratuita por Unity con el plan académico de estudiantes [32] gracias a la Universidad Autónoma de Madrid.

3.3. ML-Agents

Unity Machine Learning Agents (ML-Agents) [33] [34] es una librería de Unity de código abierto que permite trabajar con herramientas y algoritmos de aprendizaje automático con el propósito principal de entrenar agentes mediante el uso de aprendizaje por imitación, aprendizaje por refuerzo, neuroevolución, etc. Está basada en una API de Python y usa una biblioteca de TensorFlow, con las que genera redes neuronales que determinan los modelos a entrenar.

El proceso que se lleva a cabo a través de ML-Agents para el entrenamiento de modelos se puede

definir utilizando el esquema de comportamiento del aprendizaje automático (ver Figura 2.1), donde un agente recibirá información del entorno (a través de lo que ML-Agents define como “observaciones”) y utiliza esta información para tomar decisiones que traducirá en las acciones que realizará el agente.

El agente está recibiendo información de manera constante a través de observaciones, pudiendo ser estas valores o magnitudes como la velocidad del agente, la posición en forma de vector de tres coordenadas, el tiempo desde que el episodio de evaluación comienza, la distancia entre un objeto y el agente, etc. Todo esto se traspasa al modelo de forma directa, y es el propio modelo el que evalúa la información.

En ML-Agents, se puede elegir aplicar aprendizaje por refuerzo o aprendizaje por imitación. De entre los métodos de aprendizaje por imitación, destacamos dos tipos:

- *Behavioural Cloning* (Clonación de Comportamiento): Hay dos tipos, *online* u *offline*. Éstos métodos se asemejan mucho al aprendizaje supervisado, puesto que basan el comportamiento del agente en una demostración proporcionada por el desarrollador. En las demostraciones se comparten diferentes observaciones (que actúan como atributos de instancias) a la par que las tareas que realiza el agente (que sirve como resultados de las instancias). En el *Online Behavioural Cloning* la demostración se comparte en tiempo real, mientras que en el *Offline Behavioural Cloning* la demostración ha sido grabada previamente.
- GAIL (*Generative Adversarial Imitation Learning*, Aprendizaje por Imitación Generativo Antagónico): es un algoritmo basado en las *Generative Adversarial Networks* (GAN) o Redes Generativas Antagónicas. Se trata de un algoritmo con dos redes neuronales que compiten entre sí. Una de las redes es denominada “red generativa” y se encarga de generar respuestas similares a las de una base de datos mientras que la otra, la “red discriminativa”, debe detectar si la respuesta proviene de la base de datos o de la red generativa. En GAIL se utiliza ese principio, donde un agente genera acciones similares a las del desarrollador contra un discriminador que debe detectar si la acción proviene del desarrollador o del agente. El algoritmo GAIL también posee un sistema de recompensas como en el aprendizaje por refuerzo, donde al agente se le recompensa en función de las acciones que consiga hacer pasar como si fuesen del desarrollador.

No obstante, este proyecto se centrará en las posibilidades que nos ofrece ML-Agents con el aprendizaje por refuerzo, ya que los algoritmos que se utilizan en esta librería son punteros en el campo del aprendizaje automático y han demostrado resultados satisfactorios [35] [36] en comparación con otros algoritmos de aprendizaje por refuerzo.

3.3.1. Aprendizaje por Refuerzo con ML-Agents

La librería ML-Agents presenta un sistema de recompensas con el que se puede premiar las acciones correctas de nuestro agente o castigarlas en el caso de que las acciones sean incorrectas, todo ello mediante lo que se denomina como *reward signal* o señal de recompensa, que adquiere un valor numérico positivo si la acción es correcta o negativo si la acción es incorrecta.

Gracias a este tipo de sistema, se logra fomentar las acciones que forman parte de un comportamiento correcto y se suprimen las que no, condicionando al agente a cumplir las tareas propuestas. Forma parte del desarrollador determinar las recompensas, es decir, elegir qué acciones serán pre-

miadas y cuales serán penalizadas y la cantidad de recompensa por cada acción.

El sistema de recompensas está condicionado por los episodios en los que se encuentra el agente. Se entiende como episodio un intento de resolución de una prueba, que tiene un inicio en el que se restablecen e inicializan todos los elementos de la prueba (como pueden ser la localización del robot, el instanciamiento de elementos externos, etc.) y un fin determinado por varias posibilidades (como un límite de tiempo alcanzado, una acción incorrecta cometida, etc.). El sistema de recompensas utiliza la configuración de episodios para gestionar las recompensas, permitiendo al agente conocer sus resultados anteriores para considerar sus acciones y poder progresar en sus futuras pruebas.

A continuación se introducen los dos algoritmos propuestos por ML-Agents para la resolución de problemas de aprendizaje por refuerzo.

Proximal Policy Optimization

Proximal Policy Optimization (PPO), Optimización de Política Proximal en castellano, es un algoritmo desarrollado por OpenIA [37] que ha demostrado su eficiencia en múltiples campos desde la resolución de videojuegos en consolas ATARI [38] hasta el control de robots [35].

Uno de los problemas que tiene el aprendizaje por refuerzo es que el conjunto de datos de entrenamiento es dependiente de la política actual que esté aplicando el agente, ya que este genera su propio conjunto de datos de entrenamiento al interactuar con el entorno, en vez de trabajar con un conjunto de datos estático como ocurre en el aprendizaje supervisado. Esto significa que la distribución de datos sobre las observaciones y las recompensas está en constante cambio, siendo esta la causa mayor de inestabilidad en el proceso de entrenamiento.

En el aprendizaje por refuerzo se utilizan métodos de gradiente de políticas (entendemos por política como la forma de comportarse del agente en un momento dado), los cuales predicen qué camino deben seguir las políticas para mejorar sus resultados aplicando el descenso del gradiente en las redes neuronales. Pero debido a la inestabilidad comentada anteriormente es muy fácil que esta progresión de la mano de los métodos de gradiente de políticas rápidamente se desvíe y produzca resultados ineficientes o erróneos, deformando la política del agente si se aplica el descenso de gradiente en repetidas ocasiones.

Fue propuesta una solución a este problema por parte de [39], donde se plantea la idea de que si se actualiza una política, esta no podrá alejarse demasiado de la política antigua. Esta solución se conoce como *Trust Region Policy Optimization* (TRPO), y es en la que está basada PPO. El problema de TRPO es que, para restringir lo lejos que se puede desplazar una política en su evolución, implementa una serie de añadidos que aumentan la complejidad de computación, afectando al comportamiento de aprendizaje de los agentes.

PPO soluciona esto añadiendo la restricción directamente en lo que se ha definido como la función

central de optimización. A continuación, se realizará una breve explicación sobre el funcionamiento de PPO, detallada en el estudio [40].

Primero se define la variable $r_t(\theta)$ como la proporción de probabilidad entre los resultados de la nueva política actualizada y los resultados de las versiones anteriores de la red de políticas

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{old}}\left(a_t \mid s_t\right)} \tag{3.1}$$

donde a_t son las acciones tomadas por el agente y s_t el estado en el que se encuentra al tomar la acción.

Por lo tanto, dada una muestra de acciones y estados, el valor de $r_t(\theta)$ será mayor que 1 si la acción es más probable ahora que en la versión anterior de la política, u obtendrá un valor entre 0 y 1 si la acción es menos probable.

La función de avance A_t se utiliza para intentar estimar el valor relativo de la acción seleccionada, y se define como la diferencia entre la suma de recompensas del agente en cada pasa durante el episodio $\sum_{k=0}^{\infty} \gamma^k r_{t+k}$ y la función de valor $V(s)$, que se encarga de realizar una estimación de la suma de recompensas del agente desde el momento actual hasta acabar el episodio. Esta última función de valor se actualiza constantemente mediante aprendizaje supervisado, puesto que compara sus resultados con los resultados reales obtenidos al acabar el episodio.

$$A_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} - V(s) \tag{3.2}$$

Si se multiplica $r_t(\theta)$ por la función de avance A_t, se obtiene el objetivo de TRPO [39] de manera más comprensible.

$$L^{CPI} = E_t\left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{old}}\left(a_t \mid s_t\right)} A_t\right] = E_t\left[r_t(\theta) A_t\right] \tag{3.3}$$

Con esta notación se puede entonces escribir la función central de optimización de PPO:

$$L^{CLIP}(\theta) = E_t\left[min\left(r_t(\theta) A_t,\ clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) A_t\right)\right] \tag{3.4}$$

Esta función será utilizada para calcular conjuntos de trayectorias sobre el mínimo de dos términos. El primero es el objetivo de los gradientes de políticas normales, que fuerza a la política hacia acciones que producen un mejor resultado que la política base. El segundo término es una variante del primer término que contiene una versión sesgada de $r_t(\theta)$ aplicando $1-\epsilon$, $1+\epsilon$ donde ϵ es una variable que

adquiere valores muy pequeños.

Se adjunta el pseudocódigo del algoritmo PPO en el apéndice B.

El estudio [40] explica con mucho detalle todo lo expuesto anteriormente, y continúa explicando el impacto de los valores que puede adquirir la función de avance A_t entre otras muchas cosas. En este proyecto no se profundizará más, ya que su intención es únicamente introducir el algoritmo de PPO y realizar una breve explicación de su funcionamiento.

Soft Actor-Critic

Los métodos *actor-critic* son un paradigma del aprendizaje por refuerzo donde dos redes neuronales son usadas para el entrenamiento del modelo. La primera hace de actor y es quien manipula al agente, y la segunda es el crítico cuya tarea es la de evaluar la toma de decisiones del actor. Estos métodos destacan por actualizar las políticas por cada decisión del actor en vez de esperar a evaluar el resultado final de un episodio [41].

Soft actor-critic (SAC) es un algoritmo del tipo actor-critic basado en el aprendizaje por refuerzo de entropía máxima. Es decir, obtener la recompensa máxima a partir de promover políticas cuyas acciones sean lo más aleatorias posibles.

Se considera el objetivo de entropía como

$$J(\pi) = E_{\pi}\left[\sum_{t} r(s_t, a_t) - \alpha \log(\pi(a_t \mid s_t))\right] \tag{3.5}$$

donde s_t es el estado y a_t la acción. De esta forma, la política maximiza el rendimiento esperado a la par que la entropía esperada. El parámetro de temperatura no negativo α controla la compensación entre estos dos valores [42].

En el estudio [43] se explica en mayor detalle el funcionamiento del algoritmo SAC y varias de sus posibles aplicaciones, además de un esquema del pseudocódigo del propio algoritmo, que se adjunta en el apéndice B.

4

Desarrollo

En este capítulo se observará la trayectoria que ha seguido el proyecto desde su planteamiento hasta el perfeccionamiento de los entornos a los que se ha expuesto la simulación del robot. Entre estos pasos se encuentran el establecimiento de programas y entornos para el desarrollo del proyecto, la selección de elementos y parámetros utilizados y las pruebas con éstos. Por último se podrá percibir la evolución y el desarrollo del modelo establecido, cada vez más complejo.

Se adjunta el código del proyecto en el repositorio [44].

4.1. Primeros Pasos con Unity

Los inicios del proyecto comenzaron con la introducción al entorno de desarrollo Unity, del cual hemos hablado anteriormente en el apartado 3.2. Nos descargamos el programa a través de su página web [31] y realizamos un par de pruebas simplemente para familiarizarnos con los siguientes conceptos:

- *Rigidbody*: componente que permite que un objeto reaccione a físicas en tiempo real, como reacciones a las fuerzas y la gravedad, la masa o la resistencia.
- *Collider*: componente que define la forma de un objeto a efectos de colisiones físicas, sin mantener necesariamente la misma forma que el objeto en cuestión.
- *Velocity*, *angularVelocity*: vectores de velocidad y velocidad angular del componente *rigidbody*.

El material de la asignatura “Introducción a la Programación de Videojuegos y Gráficos” de la Escuela Politécnica Superior se puede utilizar para familiarizarse con el entorno de Unity.

Después de un primer contacto con Unity y sus conceptos más relevantes, se da paso al trabajo y desarrollo con la librería ML-Agents.

4.2. Introducción a ML-Agents

En este apartado se realizará una breve observación sobre la toma de contacto con la librería ML-Agents, incluyendo un repaso de la instalación de esta en Unity, la comprensión de sus elementos más

básicos y hasta la creación de un entorno de aprendizaje desde cero.

4.2.1. Instalación y Comprensión de la Librería

ML-Agents incluye una gran cantidad de documentación en la plataforma GitHub para poder comprender y habituarse a los conceptos de aprendizaje automático que conforman esta librería. Para empezar a trabajar con ella, nos dirigimos a la Guía de Inicio [45], donde aprenderemos a instalar el *Paquete ML-Agents* siguiendo una serie de instrucciones.

Más adelante, se nos introducirá a los entornos de Unity, al concepto de "agente", y se nos explicará como ejecutar un modelo pre-entrenado, de entre los muchos ejemplos ya desarrollados que están incluidos en el propio paquete. Con ello podremos entender de manera más sencilla cómo funcionan los parámetros que condicionan el comportamiento del modelo.

Por último se nos explicará como proceder para crear y entrenar un nuevo modelo utilizando aprendizaje por refuerzo, y como podremos observar y monitorizar este entrenamiento utilizando la API de Python y TensorBoard, un *kit* de herramientas de visualización de TensoFlow para poder analizar y comprender el proceso de aprendizaje de nuestro modelo.

4.2.2. Creación de un Nuevo Entorno

Una vez hemos aprendido a crear nuevos modelos de entrenamiento, a trabajar con ellos, configurarlos y entrenarlos, el siguiente paso es desarrollar un entorno completamente nuevo.

La idea inicial del proyecto era poder simular laberintos o escenarios que contuviesen plataformas o peanas de un color destacado (como puede ser rojo o verde) que se diferenciase del color del suelo. Así, el robot podría rastrear el objetivo identificando el color del suelo hasta encontrar el color del objetivo una vez estuviese posicionado encima de él. Por ello, era necesario comenzar desarrollando un entorno que contuviese un agente, que simularía al robot, y un objetivo físico, que simularía a la peana a alcanzar.

Afortunadamente, ML-Agents dispone de una guía [46] para desarrollar un entorno de aprendizaje que contiene estos dos elementos, y que nos sirve de punto de partida para empezar a desarrollar nuestros entornos de aprendizaje del proyecto.

Este nuevo entorno de aprendizaje consiste en un agente esférico que debe alcanzar un objetivo con forma de cubo dentro de una plataforma (ver figura 4.1). El agente puede caerse por la plataforma, ya que esta no tiene bordes. Si esto ocurriese, se reiniciaría el entorno y el agente sería penalizado. Al entrenar este agente, se puede observar cómo desarrolla pautas de movimientos donde cada vez es más eficiente a la hora de alcanzar el objetivo, y mantiene una conducta restrictiva en cuanto se acerca a los límites de la plataforma.

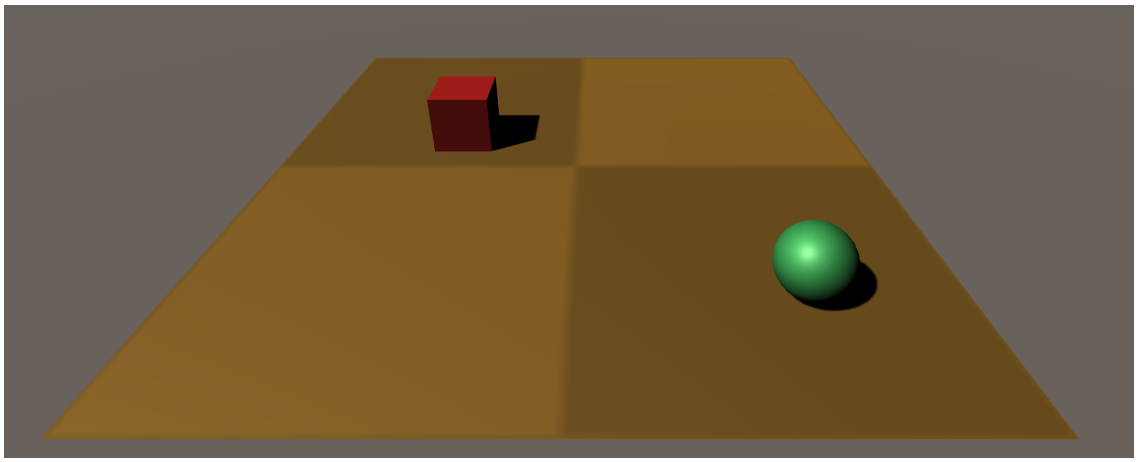


Figura 4.1: Entorno de aprendizaje, ejemplo propuesto por ML-Agents

En la guía se nos explica primero cómo crear un proyecto nuevo en Unity (instalando de nuevo el Paquete ML-Agents). Después, procede a explicarnos cómo crear el entorno físico, incluyendo un plano que actúa como suelo, una esfera que hará de agente y un cubo que actuará de objetivo.

En este paso, también se nos explica como programar el comportamiento del agente y del propio entorno (puesto que este puede cambiar también) a través de un *script* (en Unity se pueden utilizar varios lenguajes de programación, pero nosotros trabajaremos con *C#* que es el que utilizan en ML-Agents por defecto). Con este script nos explicarán cómo inicializar y resetear el entorno, cómo recolectar la información del entorno que percibe el agente y compartirla con el modelo, cómo compartir con el agente las posibles acciones producidas por el modelo, y como asignar y gestionar las recompensas del modelo.

A continuación, se nos enseñará a establecer los parámetros del agente, que incluye la configuración de los vectores de observación, determinar el tipo de acciones que se utilizan en el aprendizaje, la selección de parámetros e hiperparámetros de comportamiento, etc.

Por último, la guía nos muestra como probar el entorno, cómo utilizarlo para entrenar modelos y cómo realizar un seguimiento del aprendizaje de éstos, de una manera parecida a la Guía de Inicio comentada en la sección anterior.

4.3. Medición del Robot

Nuestro siguiente paso es sustituir al agente esférico por un modelo simulado de nuestro robot NXT de LEGO® MINDSTORMS®, y para ello necesitamos saber las características de este. Procedemos midiendo sus dimensiones y su peso, pero también necesitamos saber sus posibles configuraciones de velocidad y giros.

Se ha decidido implementar una configuración de movimiento basada en la combinación de maniobras predefinidas en contraposición a una configuración de movimiento continuo, ya que así se facilita en gran medida el trabajo con las redes neuronales. En consecuencia, se han obtenido muy buenos resultados aplicando esta técnica.

A través del entorno de desarrollo Bricx Command Center, se establecen una serie de procedimientos para probar los posibles movimientos del robot y así poder elegir cuales son las mejores configuraciones de potencia de los servomotores que dictaminarán su velocidad y el tiempo de los giros. En el apéndice C se incluye un ejemplo de procedimiento del programa Bricx Command Center.

Se han probado múltiples combinaciones de velocidades, midiendo las distancias en función de los segundos que el robot se desplazaba. También se han medido los ángulos de rotación en función de los segundos para seleccionar los valores óptimos de giro. Con todo ello, se han determinado los valores de la potencia de los servomotores para establecer un estándar en los movimientos del robot. Se adjunta la tabla de mediciones en el apéndice D.

A continuación, se puede observar en el cuadro 4.1 los porcentajes de velocidad seleccionados para cada tipo de movimiento. Las medidas están establecidas de manera que se consideran valores positivos (en las magnitudes de velocidad en cm/s y giro en rad/s) para avanzar y girar a la derecha, y valores negativos para retroceder y girar a la izquierda.

Tipo de movimiento	Pot. motor der.	Pot. motor izq.	Velocidad	Giro
Robot estacionado	0	0	0 cm/s	0 rad/s
Avance	100	100	24.7 cm/s	0 rad/s
Avance y giro a la derecha	45.28	100	12.35 cm/s	0.52 rad/s
Avance y giro a la izquierda	100	43.34	12.35 cm/s	-0.52 rad/s
Retroceso	-100	-100	-24.7 cm/s	0 rad/s
Retroceso y giro a la derecha	-45.28	-100	-12.35 cm/s	-0.52 rad/s
Retroceso y giro a la izquierda	-100	-43.34	-12.35 cm/s	0.52 rad/s
Rotación a la derecha	-85	85	0 cm/s	1.57 rad/s
Rotación a la derecha	85	-85	0 cm/s	-1.57 rad/s

Tabla 4.1: Potencias de los servomotores para cada tipo de movimiento que efectua el robot real

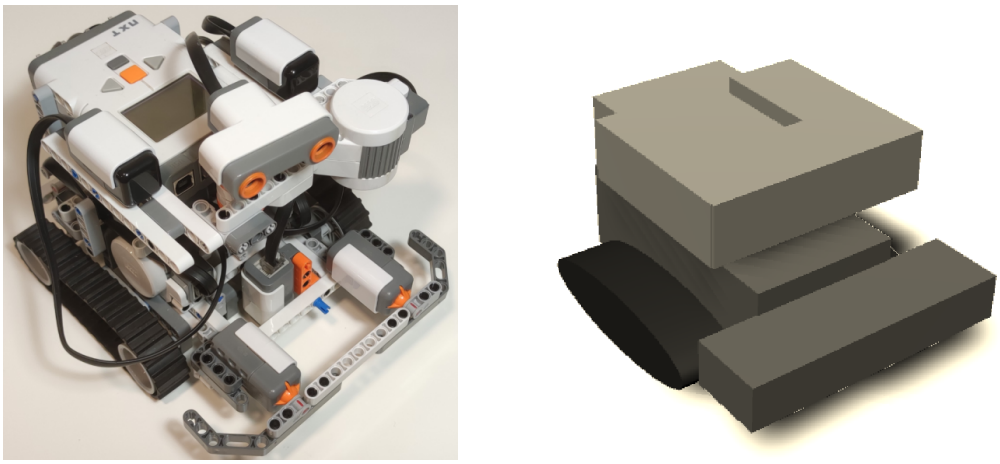
Los valores de los giros se han establecido así ya que con esos valores se logran fracciones de giro a partir de valores enteros. Es decir, si se realiza un giro a la izquierda o derecha durante 3 segundos se logrará girar 90 grados, y si se decide rotar hacia izquierda o derecha durante 2 segundos se logra una rotación de 180 grados. Ambos valores son especialmente útiles para mantener una rigurosidad a la hora de girar o rotar.

El robot pesa 940 gramos aproximadamente, así que se ha establecido que su peso en la simulación sería de 1 Kg para satisfacer de forma más sencilla las físicas de Unity, y sus dimensiones son 17

cm de alto, 22 cm de largo y 16 cm de ancho.

4.4. Adaptación del Robot Real a Unity

Ahora que ya sabemos las características del robot, nuestro objetivo será el de crear un modelo en Unity que reúna estas características. Para mejorar la experiencia, se ha diseñado un modelo que simula la forma del robot real, incluyendo el cuerpo del robot, el parachoques frontal y las ruedas, como se puede observar en la figura 4.2, aunque la apariencia del agente no es realmente relevante para el entrenamiento de este.



(a) Fotografía del robot real

(b) Simulación del robot en Unity

Figura 4.2: Comparación entre los modelos del robot real y el robot simulado.

Se ha determinado la configuración del movimiento del agente y el aprendizaje del modelo de manera que se logre una mayor adaptabilidad de cara al futuro, de la siguiente manera (ver figura 4.3):

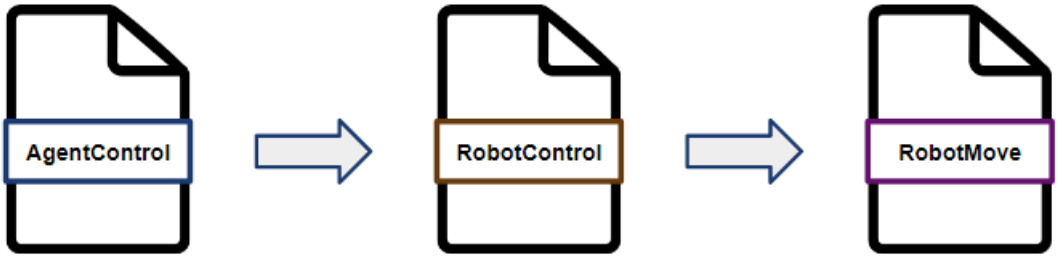


Figura 4.3: Diagrama de archivos de configuración de movimiento y control del agente

Primero se debe comprender que el movimiento del robot está dictaminado por dos variables llamadas “acciones” (*Action1*, *Action2*), las cuales toman valores discretos entre 0 y 2. (la explicación de por qué esto se ha decidido así se puede encontrar en el apartado 4.5.2). La primera variable dicta si

el robot avanza (*Action1* = 2), retrocede (*Action1* = 0) o se mantiene estático (*Action1* =1), mientras que la segunda variable se utiliza para rotar al robot a la derecha (*Action2* = 2), izquierda (*Action2* = 0) o para no rotar (*Action2* = 1). Las variables "acciones" son las salidas que proporciona la red neuronal que se está entrenando, las cuales son gestionadas a través del archivo *AgentControl.cs* (que se encarga del comportamiento del agente), y a través de una serie de *scripts* se traducen estos valores en movimientos que la simulación ejecuta.

Se ha creado un paso intermedio en este proceso de traducción, una pequeña interfaz que en un futuro se utilizaría para controlar los movimientos del robot real además de la simulación. Esto se encuentra en el archivo *RobotControl.cs*, el cual recibe el valor de las variables "acciones" del modelo de aprendizaje gracias a *AgentControl.cs*. En función de la combinación de estos valores, *RobotControl.cs* guarda en otras dos variables (*rleft*, *rright*) la potencia que deberían establecer los servomotores del robot real, justo como se ha hecho anteriormente a través de Bricx Command Center.

Por último, se comparten las variables de potencia de los servomotores establecidas en *RobotControl.cs* con el archivo *RobotMove.cs*, que se encarga, ahora sí, de configurar el movimiento del robot simulado. La utilidad de esta combinación de ficheros permite fácilmente adaptar cualquier programa futuro que tenga la intención de controlar al robot real, en cuyo caso recibiría la información directa de *RobotControl.cs*, o si se quisiese adaptar instrucciones del robot real a una simulación, que se compartirían directamente con *RobotMove.cs*.

4.5. Configuración de la Simulación del Robot

Se han realizado múltiples pruebas con la simulación del robot con el objetivo de adaptar el movimiento y las observaciones de este. Además, se han tomado distintas decisiones sobre si utilizar acciones continuas o discretas y sobre la configuración del sistema de recompensas del modelo.

A continuación, se comentará la evolución del proyecto hasta su estado actual junto con los puntos más relevantes del desarrollo del modelo.

4.5.1. Configuración del Movimiento

Los primeros pasos tras la implementación del robot simulado han consistido en lograr que el movimiento del robot fuese lo más parecido al del robot real. Se ha logrado que el robot se moviera de manera realista utilizando las funciones *AddForce* y *AddTorque* del componente *rigidbody*, logrando que la simulación avanzase a 24.7 cm/s, que girase con una velocidad angular de 0.52 rad/s y que rotase sobre sí mismo a 1.57 rad/s.

4.5.2. Selección de Acciones Continuas o Discretas

Uno de los factores que se ha tenido en cuenta ha sido elegir entre las dos posibilidades que ofrece la librería ML-Agents: si utilizar vectores de acciones discretas o continuas. Las acciones continuas toman valores numéricos con decimales entre -1 y 1 (aunque esto puede ser modificado por el desarrollador), útiles para designar valores como la velocidad o la aceleración del agente, mientras que las discretas toman valores numéricos enteros entre intervalos establecidos por el desarrollador.

Se han realizado pruebas con ambas opciones y los resultados han sido muy similares, pero el uso de acciones continuas requiere de mayor computación y por lo tanto requiere de un mayor tiempo de entrenamiento (ver apartado 5.1.2). Por ello, se ha decidido utilizar las acciones discretas, ya que se determinó desde un principio establecer movimientos predefinidos como se ha visto en el apartado 4.3, y puesto que su implementación es más directa y sencilla para las redes neuronales al no tener que trabajar con intervalos de valores como ocurre si se eligieran las acciones continuas, aunque debido a su gran capacidad de cómputo no existe una diferencia sustancial en nuestras pruebas.

4.5.3. Evolución del Entorno y el Objetivo

Si se recuerda lo comentado en el apartado 4.2, el entorno de aprendizaje se ha basado en otro entorno propuesto por ML-Agents, el cual ha sido modificado y adaptado hasta conseguir un entorno que pudiese satisfacer los objetivos propuestos en el Trabajo de Fin de Grado.

Para el entrenamiento del agente se utilizarán las herramientas de aprendizaje automático contenidas en el subpaquete de Python incluido en el *Paquete ML-Agents*, el cual contiene el conjunto de algoritmos y redes neuronales que formarán los modelos que queremos entrenar. Se han probado los dos algoritmos proporcionados por ML-Agents, PPO y SAC (ver apartado 3.3.1) en los distintos entornos desarrollados durante este proyecto, de los cuales se comentará a continuación (en la apartado 5.2.2 se entrará en detalle acerca de los resultados obtenidos por cada uno de los algoritmos).

Primero se han implementado bordes en el entorno para que el agente no se cayese como ocurría en el entorno propuesto por ML-Agents, y por lo tanto se ha tenido que implementar un sistema de cuenta atrás para la resolución del propio entorno. Ahora, aunque los episodios de aprendizaje siguen dependiendo de que el agente encuentre el objetivo, éstos ya no dependen de que el agente se caiga por uno de los bordes de la plataforma (ya que ya no puede caerse), sino de que el agente encuentre el objetivo en un tiempo determinado, y en caso de superarse ese tiempo el entorno se reiniciaría.

El siguiente paso fue implementar obstáculos como pilares o paredes para ver si el agente era capaz de resolver los entornos de forma eficiente. Y en efecto, en el momento en el que el agente lograba aprender su objetivo conseguía cada vez mejores resultados.

4.5.4. Implementación de los Sensores

Las entradas que se proporcionan a la red neuronal utilizada en proyecto son suministradas a través de lo que ML-Agents denomina como “observaciones”. Al igual que con las “acciones” (se recuerda en el apartado 4.4 que estas son las salidas generadas por la red neuronal), es labor del desarrollador elegir el tamaño del vector de las observaciones. Es decir, es el desarrollador el que elige el número de entradas y salidas que tiene su modelo, todo ello fácilmente implementable gracias a la librería ML-Agents.

En el entorno de aprendizaje propuesto de ejemplo, el agente sabe en todo momento su posición y la posición que ocupa el objetivo, utilizando vectores tridimensionales (*Vector3*) proporcionados por Unity. Junto con su velocidad lineal y velocidad angular, estos datos componen las observaciones del agente.

El robot real puede saber su velocidad lineal y angular gracias a su sensor giroscópico y su sensor acelerómetro (ver figura 3.1), pero no puede saber su posición en función de unas coordenadas, y tampoco puede saber la posición del objetivo. Por ello, se han tenido que eliminar las observaciones de posición en función de las coordenadas de ambos agente y objetivo y añadir nuevas observaciones que el robot real pudiera generar. Y estas observaciones provienen de la implementación de los componentes *Raycast* de Unity y *RayPerceptionSensor* de ML-Agents:

- Se utiliza *Raycast* para la detección de elementos cercanos al agente. El primero de ellos se utiliza simulando el comportamiento del sensor de color de nuestro robot real, puesto que detecta cambios en el color del suelo. También se utiliza *Raycast* para imitar el sistema de detección de colisiones explicado anteriormente en el apartado 3.1.1, donde se detectan elementos muy cercanos al robot, representando una colisión real. Se han utilizado booleanos como observadores, indicando si se detecta o no un color o una colisión.
- *RayPerceptionSensor* es un componente prefabricado por la librería ML-Agents que permite simular de forma sencilla el sensor ultrasónico del robot real. Este componente es configurable a la hora de establecer el ángulo de detección y la distancia que alcanza el sensor, y se ha elegido un ángulo de aproximadamente 90 grados centrado al frente del modelo. El vector de observación generado por este componente se alimenta directamente en la red neuronal sin la necesidad de declararlo como se ha tenido que hacer con el resto de observaciones, lo que hace muy cómoda la implementación de este componente.

Ahora, el robot simulado es capaz de orientarse por el entorno, localizando elementos en su frontal, detectando los colores del suelo en el que se encuentra y simulando colisiones cuando se acerca demasiado a un obstáculo o pared.

4.5.5. Sistema de Recompensas

Al estar trabajando con aprendizaje por refuerzo, se debe implementar un sistema de recompensas que sea capaz de indicar al modelo si está obrando correctamente o no (esto ya se vió en el apartado 2.3).

	Observación	Tipo	Descripción
1	rBody.velocity.x	Vector3	Valor de la velocidad en el eje X medido en m/s
2	rBody.velocity.z	Vector3	Valor de la velocidad en el eje Z medido en m/s
3	fracTargetsReached	Float	Cantidad de objetivos alcanzados en función del total de objetivos
4	hittingFw	bool	Contacto frontal
5	hittingLeft	bool	Contacto lateral izquierdo
6	hittingRight	bool	Contacto lateral derecho

Tabla 4.2: Observaciones utilizadas en el modelo del proyecto

En el entorno de aprendizaje propuesto como ejemplo se reparte una recompensa positiva cuando el agente logra alcanzar el objetivo y se le penaliza con una recompensa negativa cuando se cae por la plataforma. Es un sistema sencillo y efectivo para un entorno sencillo.

Se ha adaptado este sistema a nuestro entorno de aprendizaje, ya que este es bastante más complejo que el entorno propuesto como ejemplo.

Primero se ha variado la penalización por caerse de la plataforma, ya que el agente no puede caerse debido a los bordes. Ahora, se penaliza de la misma manera que el agente no consiga resolver el entorno en el tiempo determinado.

También se han añadido recompensas para condicionar y fomentar ciertas conductas del modelo:

- Se ha añadido una muy pequeña penalización cuando el agente se mueve. Al principio esto puede ser contraintuitivo, pero después de muchas pruebas esta solución ha dado resultados muy positivos, ya que el agente giraba sin control en sus inicios del aprendizaje al no tener muy claro cuál era su objetivo. Este tipo de conductas acababan resultando en un aprendizaje lento e ineficiente. En resumen, el agente ahora mantiene una política conservadora en cuanto a realizar movimientos, puesto que si realiza muchos movimientos en vano esto resultará en una penalización mayor.
- Se ha añadido una pequeña penalización cuando el agente retrocede, levemente superior al resto de movimientos. La penalización es, aun así, muy pequeña para evitar que el robot nunca elija retroceder, pero si lo suficiente como para favorecer un movimiento frontal, ya que es donde los sensores de distancia *RayPerceptionSensor* y *Raycast* apuntan y de donde obtienen información.
- Cuando la simulación del robot se acerca a una pared, se suprime la penalización por retroceder para fomentar una corrección de la trayectoria y, sin embargo, penalizamos cualquier otro tipo de movimiento que pudiera promover una posible colisión.
- Si la simulación del robot se acercase demasiado a una pared se le penalizaría bastante más, simulando un choque frontal interceptado con el sistema de detección de colisiones del robot real.
- Por último, se han modificado las recompensas proporcionadas cuando el agente alcanza un objetivo, puesto que en el nuevo entorno es posible que existan más de uno. En el sistema actual se reparte una recompensa acumulativa, que aumenta en función del número de objetivos encontrados, hasta alcanzar el último. La recompensa total por encontrar todos los objetivos es la misma que la que se le entrega al agente en el entorno de ejemplo, lo cual significa que las recompensas por encontrar los primeros objetivos en nuestro entorno son bastante bajas,

mientras que encontrar los últimos objetivos es altamente recompensado. Con esto se fomenta que el modelo desarrolle un interés por completar el entorno en su totalidad a la vez que se logra un correcto aprendizaje en el inicio del entrenamiento del modelo.

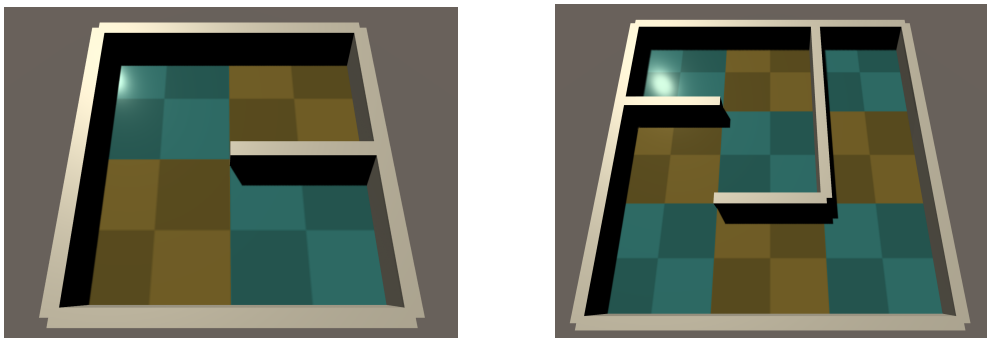
Se adjunta en el apéndice E una tabla que incluye la configuración de recompensas utilizadas.

4.5.6. Desarrollo de Laberíntos Dinámicos

Una de las características más destacables para el aprendizaje que contiene el entorno de ejemplo propuesto por ML-Agents es el hecho de que este contiene un factor de aleatoriedad. Cuando se inicializa el entorno, el agente siempre aparece en el centro de la plataforma, ya sea porque es el primer episodio del entorno o porque el agente se cayó por la plataforma y el entorno fue restablecido. Pero, en cualquiera de estos dos casos el objetivo aparece en una posición aleatoria de la plataforma (utilizando la función *random* de Unity), y también ocurre cuando el agente alcanza al objetivo (este reaparece en otra posición de la plataforma).

Esto ha sido planteado así para que el agente realmente sea capaz de aprender a alcanzar el objetivo desde cualquier posición, y no que simplemente aprenda un camino y lo repita una y otra vez.

Por ello, se han tenido que desarrollar entornos que mantuviesen una aleatoriedad similar a la del entorno de ejemplo. Se han desarrollado entornos de aprendizaje cuadrados de estilo laberíntico, semejante a la forma de una matriz de 2x2 y de 3x3 (ver figura 4.4).



(a) Laberinto de tamaño 2x2

(b) Laberinto de tamaño 3x3

Figura 4.4: Ejemplo de laberintos dinámicos generados aleatoriamente

La aleatoriedad de estos entornos laberínticos se encuentra en la disposición de las paredes. Al establecer únicamente algunas paredes de manera pseudoaleatoria (ya que esta aleatoriedad no es completa, está condicionada para no crear objetivos inalcanzables y que siempre exista un recorrido posible) se consigue implementar episodios de aprendizaje donde cada laberinto es distinto al anterior, evitando que el agente aprenda únicamente un camino, y logrando promover comportamientos de exploración. Se adjunta el código de generación de los laberintos en el apéndice F.

5 Integración, Pruebas y Resultados

En este capítulo se expondrán las pruebas con los entornos de aprendizaje que se han desarrollado a lo largo del proyecto, desde los primeros entornos para comprobar si el agente era capaz de resolverlos hallando un objetivo aplicando la nueva configuración de movimiento que imita la del robot real, hasta las últimas pruebas de exploración en entornos donde el robot no ha llegado a entrenar.

El objetivo de estas pruebas es que el agente logre aprender un comportamiento con el que consiga resolver las distintas misiones de exploración a las que se le expone.

5.1. Prueba 1: Configuración de Movimiento

El objetivo de esta prueba era el de comprobar si el agente logra resolver los entornos de aprendizaje con la nueva configuración de movimiento adaptada a partir de los movimientos predefinidos seleccionados a la hora de medir el comportamiento del robot (ver apartado 4.3).

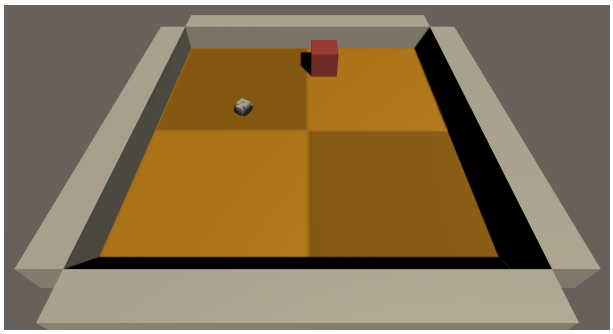


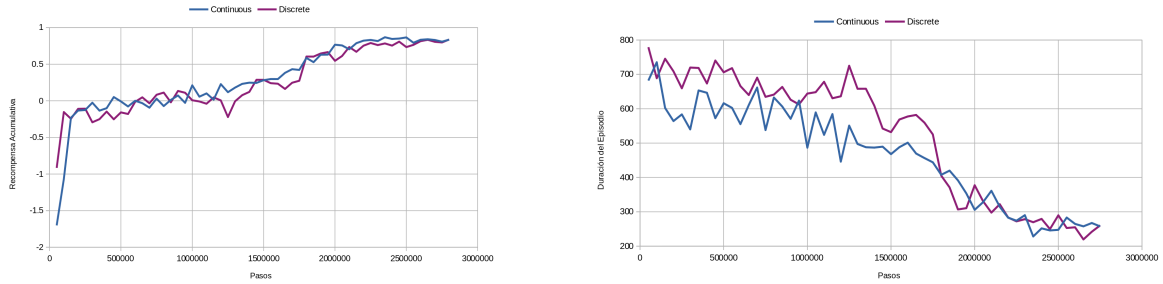
Figura 5.1: Ejemplo de entorno de aprendizaje utilizado en la Prueba 1

5.1.1. Configuraciones de la Prueba 1

Se ha utilizado una red neuronal con 2 capas ocultas y 128 unidades por cada capa. También se ha utilizado el algoritmo PPO con una configuración de hiperparámetros adjunta en el apéndice G.

Se ha comparado el uso de acciones continuas y discretas por parte de la red neuronal para comprobar si existía alguna preferencia en función de la eficiencia. También se ha utilizado aprendizaje paralelo entrenando 9 entornos a la vez en cada configuración. Los entornos entrenarán durante 2.8M de pasos (aproximadamente 1h de entrenamiento), que son la unidad que usa ML-Agents para contabilizar las acciones realizadas en el episodio.

5.1.2. Resultados Obtenidos de la Prueba 1



(a) Recompensa Acumulativa en función de los Pasos

(b) Duración del Episodio en función de los Pasos

Figura 5.2: Resultados obtenidos de la Prueba 1

Como se puede observar en las gráficas, ambas configuraciones han logrado un aprendizaje satisfactorio y bastante semejante, siendo la configuración continua la que mejores resultados ha llegado a aportar en los inicios. Sin embargo, al final del entrenamiento ambas configuraciones han alcanzado unos resultados muy parejos tanto en la recompensa acumulativa (la cantidad de recompensa que pueden completar por episodio) como en la duración de los episodios (medida en pasos).

Ambas pruebas han tenido una duración de 2.8M de pasos, pero el tiempo de entrenamiento con la configuración continua ha sido levemente mayor (1h 10m 39s) frente a la configuración discreta (59m 58s), lo cual indica que el peso de la computación de la configuración continua es mayor que el de la discreta.

La configuración continua tiende a realizar muchos más giros que la discreta y en la robótica los giros constituyen un gasto mayor de energía. Por este motivo, porque la configuración continua tiene un coste computacional mayor y porque ambas configuraciones producen resultados muy similares se ha elegido la configuración discreta como configuración por defecto para el resto de pruebas.

5.2. Prueba 2: Configuración de Sensores

En esta prueba ya se han implementado los sensores de distancia, de color y de contacto, y se ha adaptado la configuración de observación del agente, el cual ya no conoce su posición o la del objetivo

de forma implícita (ver apartado 4.5.4).

El objetivo de esta prueba es comprobar que el agente es capaz de resolver los entornos de forma eficiente con su nueva configuración, siendo capaz de detectar y esquivar obstáculos en su trayectoria. También se ha modificado al objetivo, el cual deja de ser un cubo para convertirse en una plataforma que el agente deberá encontrar cuando se encuentre encima, forzando una exploración en el entorno de aprendizaje. Por último, se probarán los dos algoritmos de aprendizaje por refuerzo que incluye ML-Agents en su librería y compararemos resultados.

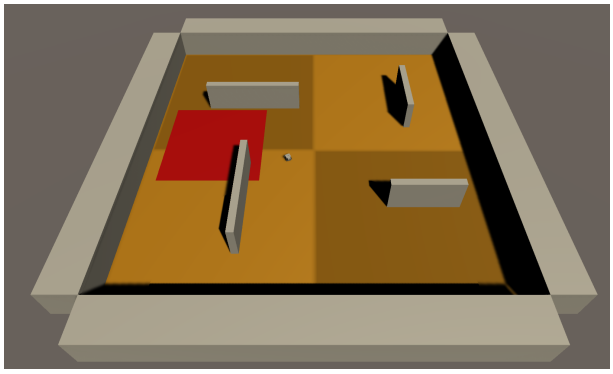


Figura 5.3: Ejemplo de entorno de aprendizaje utilizado en la Prueba 2

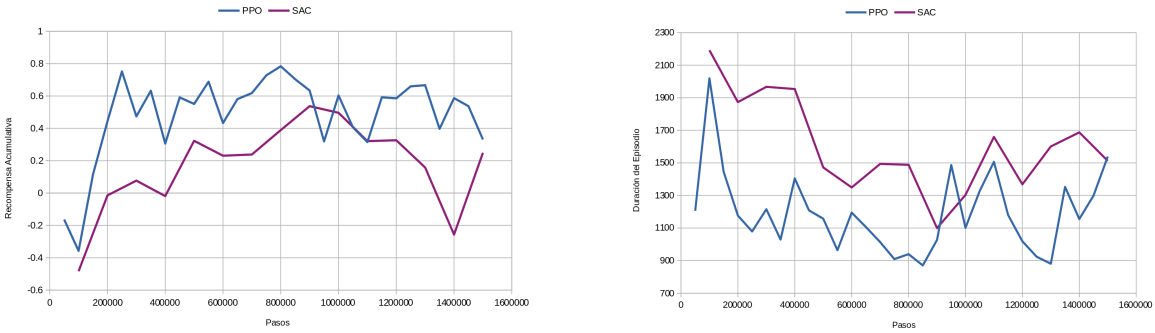
5.2.1. Configuraciones de la Prueba 2

Se han probado dos configuraciones, una con el algoritmo PPO y otra con el algoritmo SAC. Se han utilizado los parámetros por defecto recomendados por ML-Agents, adjuntados en el apéndice H. En ambas configuraciones se ha utilizado una red neuronal con 2 capas ocultas y 128 unidades por cada capa. Se recuerda que el número de entradas y salidas viene definido por las observaciones y las acciones (ver apartado 4.4), que en estas pruebas siempre son 6 entradas (observaciones) y 2 salidas (acciones). Como hemos comentado en la anterior prueba, se ha utilizado la configuración de acciones discretas en ambas pruebas. El algoritmo de SAC no ha permitido la paralelización con 9 entornos a la vez, teniendo que limitarla a 4 entornos únicamente, mientras que PPO ha podido entrenar con los 9 entornos como en la prueba anterior. Ambas pruebas han entrenado hasta conseguir 1.5 M de pasos.

5.2.2. Resultados Obtenidos de la Prueba 2

Los resultados obtenidos en esta prueba no son tan satisfactorios como en la anterior. Esto se debe a varios motivos:

- El primero y el más relevante es la complejidad del entorno. El agente en ambas configuraciones ha comprendido rápidamente qué era lo que tenía que hacer, pero aun así no lograba completar la tarea de forma eficiente.
- El segundo motivo es la modificación de los sensores. Al no conocer su propia posición y la del objetivo, el agente ahora debe explorar de forma ciega hasta encontrar el objetivo.



(a) Recompensa Acumulativa en función de los Pasos

(b) Duración del Episodio en función de los Pasos

Figura 5.4: Resultados obtenidos de la Prueba 2

Al no poder entrenar con una paralelización mayor, SAC ha obtenido sus resultados a costa de mayor tiempo de entrenamiento (1h 7m 22s) frente a PPO que ha logrado su objetivo en prácticamente la mitad de tiempo (35m 54s). Los resultados obtenidos por ambos son similares, estando PPO por encima de SAC en los inicios del entrenamiento. Por lo mencionado anteriormente, se ha decidido seleccionar a PPO como algoritmo para realizar las siguientes pruebas.

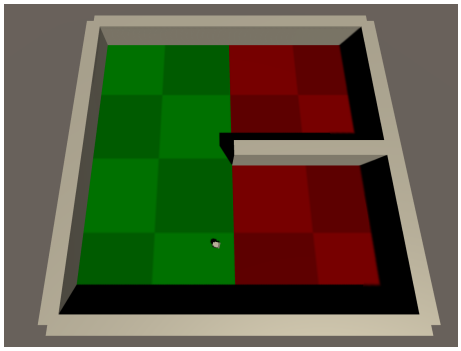
5.3. Pruebas 3.1 y 3.2: Laberintos Dinámicos

En este tipo de pruebas se han implementado los entornos dinámicos mencionados en el apartado 4.5.6, donde cada episodio configura un laberinto distinto. El objetivo de estas pruebas es comprobar que el agente consigue resolver los laberintos independientemente de su estructura, constatando que el aprendizaje es completo y que no se limita simplemente a memorizar un único trayecto.

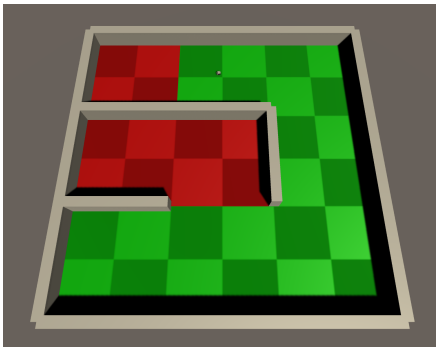
También se ha cambiado la configuración de objetivos, siendo estos más accesibles para el agente con el fin de que puede relacionar mejor cuál es su objetivo, evitando así un aprendizaje poco fructífero como el ocurrido en la prueba anterior. En esta prueba existen múltiples objetivos (uno por bloque dentro del laberinto), y el agente es recompensado cada vez que descubre un objetivo nuevo. Se han representado los objetivos nuevos con el color rojo y los objetivos ya descubiertos con el color verde (ver figura 5.5).

5.3.1. Configuraciones de las Pruebas 3.1 y 3.2

Los laberintos de dimensión 2x2 son relativamente sencillos, mientras que los laberintos 3x3 constituyen un reto mayor para el agente. Por ello, se utilizarán las pruebas con los laberintos 2x2 para observar el proceso de aprendizaje, comparando distintos hiperparámetros (se adjunta la configuración de estos en el apéndice I) con el objetivo de encontrar la configuración más eficiente posible. La



(a) Ejemplo de laberinto de dimensiones 2x2



(b) Ejemplo de laberinto de dimensiones 3x3

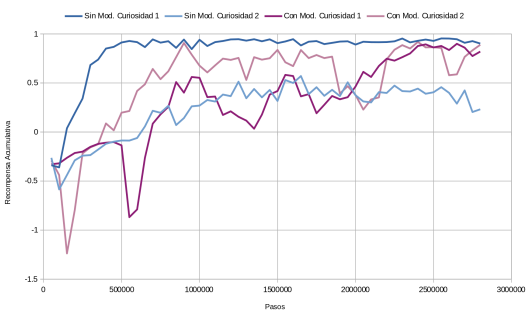
Figura 5.5: Ejemplos de entornos de aprendizaje utilizados en las Pruebas 3.1 y 3.2

primera configuración consta de los hiperparámetros propuestos por defecto en ML-Agents al usar el algoritmo PPO, mientras que la segunda configuración incluye un módulo de curiosidad recomendado por ML-Agents para algunos tipos de entornos de aprendizaje. El objetivo de este módulo preconfigurado por ML-Agents es el de incentivar los comportamientos que promuevan la exploración.

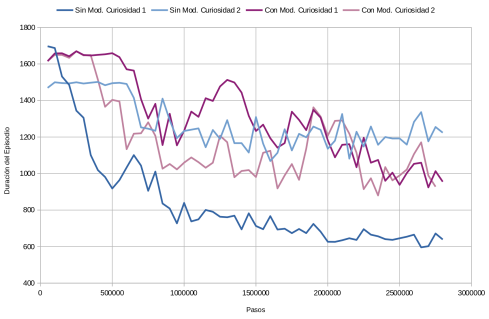
Ambas pruebas constan de una red neuronal con 2 capas ocultas y 128 unidades por capa, el tiempo de resolución máximo de los episodios es de 20 segundos y se entrenará cada modelo hasta un total de 2.8M de pasos (aproximadamente 1h).

Para las pruebas con laberintos de 3x3 también se han comparado configuraciones con el módulo curiosidad. Se compararán tres pruebas donde cambiará la configuración de las redes neuronales utilizadas, siendo la primera prueba la que contenga una red neuronal de 2 capas ocultas con 128 unidades por capa, mientras que la segunda y la tercera contendrán una red neuronal con 3 capas y 256 unidades por capa. Se adjunta la configuración de los hiperparámetros en el apéndice I.

5.3.2. Resultados Obtenidos de las Pruebas 3.1 y 3.2



(a) Recompensa Acumulativa en función de los Pasos

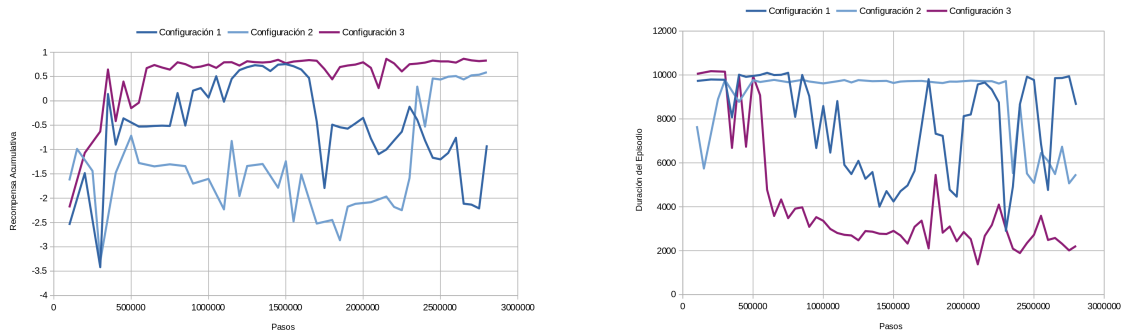


(b) Duración del Episodio en función de los Pasos

Figura 5.6: Resultados obtenidos de la Prueba 3.1

En la figura 5.6 se pueden observar dos comportamientos destacables. El primero es que en los casos en los que no se usa el módulo de curiosidad el agente puede lograr resultados muy positivos o en su defecto resultados de media eficacia. Esta irregularidad se debe a la aleatoriedad del entrenamiento en sus inicios. Es decir, si el agente encuentra una estrategia positiva en sus primeros pasos, este la maximizará obteniendo resultados muy positivos. Por otro lado, si no logra dar con una estrategia óptima, puede desarrollar un comportamiento que no será tan eficiente y que tenga dificultad para mejorar sus resultados. Con un correcto entrenamiento y con más tiempo de práctica, se ha observado que el agente acaba resolviendo el ejercicio cada vez mejor.

El segundo comportamiento se encuentra en las configuraciones con el módulo curiosidad, las cuales tienen a lograr resolver el laberinto de forma eficiente, pero en su trayectoria de entrenamiento tienen lugar cambios de estrategia que perjudican la correcta resolución, y aunque acaben siendo descartados influyen en el entrenamiento de forma negativa, haciendo que este sea más lento. Estos entrenamientos dan mejores resultados que los resultados medios de la configuración sin el módulo pero sin llegar a ser tan eficientes como los mejores resultados logrados.



(a) Recompensa Acumulativa en función de los Pasos

(b) Duración del Episodio en función de los Pasos

Figura 5.7: Resultados obtenidos de la Prueba 3.2

Como se puede ver en la figura 5.7, se ha probado el módulo curiosidad en los laberintos de dimensión 3x3 con tres configuraciones que contienen distintas redes neuronales como se ha comentado previamente. La primera configuración contiene una red neuronal de 2 capas ocultas de 128 unidades. Esta ha logrado resolver los laberintos, pero con el paso del tiempo y debido al módulo curiosidad su estrategia ha sido tergiversada y ha empezado a producir resultados ineficientes. Y es que la resolución de estos laberintos es bastante compleja, los agentes deben hallar cuál es el objetivo y esto es difícil en muchos casos, mostrando una tendencia al desaprendizaje.

La segunda configuración se ha probado con una red neuronal con 3 capas ocultas y 256 unidades por capa. Esto quiere decir que el entrenamiento será más lento pero con mayor capacidad y profundidad de aprendizaje. Como se puede observar, el agente tarda más en comprender el objetivo que debe lograr, pero cuando lo consigue empieza a producir resultados bastante positivos. Aun así el módulo curiosidad hace que el entrenamiento sea más complicado y lento. variando las estrategias

con el objetivo de encontrar mejores opciones pero sin lograr resultados.

Por último se ha probado la tercera configuración con una red neuronal que contenía 3 capas ocultas y 256 unidades por capa, pero sin incluir el módulo curiosidad. Esta configuración ha logrado comprender el objetivo de forma relativamente rápida, y eso se ha manifestado en un aprendizaje muy eficiente, superando ampliamente los resultados de las dos configuraciones anteriores.

5.4. Prueba 4: Resolución de Laberintos Mayores

Se ha decido probar el modelo ya entrenado de la última configuración de las pruebas de laberintos de dimensiones 3x3 con laberintos de una mayor dimensión, con el objetivo de comprobar si el modelo ha aprendido a resolver laberintos en general, o por el contrario, si el modelo únicamente es capaz de resolver laberintos con la dimensión entrenada.

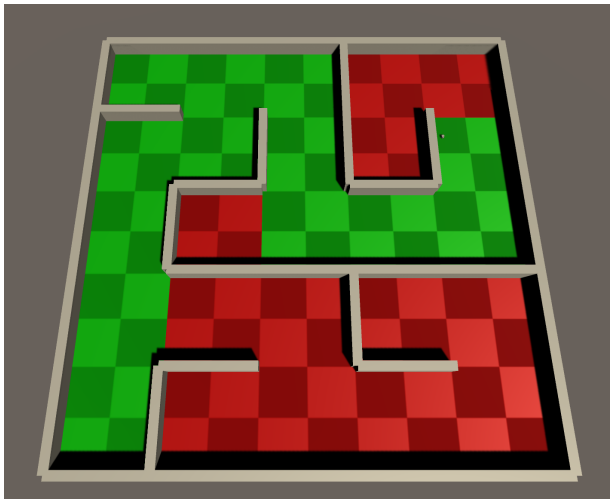


Figura 5.8: Entorno de aprendizaje utilizado en la Prueba 4, un laberinto de dimensión 5x5

Se ha puesto en prueba el modelo con el laberinto que se puede observar en la figura 5.8, un laberinto de dimensión 5x5 con varios callejones. Esta vez no se ha entrenado el modelo en el laberinto, sino que se ha utilizado un modelo que ya ha completado su fase de entrenamiento.

Se puede encontrar el video con la resolución del laberinto y demás entrenamientos en [47]:

Los resultados obtenidos son muy positivos, el agente ha logrado resolver el laberinto en un tiempo proporcional al tamaño del laberinto (14m 30s) en función a los tiempos de entrenamiento con laberintos de dimensión 3x3 (aproximadamente 3m 45s), pasando de 9 a 25 objetivos que debe encontrar para resolver el laberinto.

Se observa con mayor claridad que el agente ha desarrollado por su cuenta una estrategia similar a la “Regla de la Mano Derecha”, donde el sujeto recorre el laberinto sin apartar su mano derecha (o lateral derecho) de la pared que tiene a la derecha, manteniendo esta regla en todo momento

incluyendo los giros que realiza el sujeto. En este caso, el agente ha optado por mantenerse cerca de la pared izquierda en todo momento, y podemos observar como encuentra las plataformas objetivo en este orden justamente.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

El proyecto comenzó con la familiarización del entorno de desarrollo Unity y la librería ML-Agents, con la intención de comprender y utilizar estas herramientas para poder implementar la simulación del robot en entornos de aprendizaje.

Después se analizó el robot NXT y se hizo un estudio y recopilación de sus atributos y posibles configuraciones para lograr una implementación realista.

Se desarrollaron entornos de aprendizaje donde se introdujo la configuración del robot al agente dentro de esos entornos, y se probaron las configuraciones para verificar que el agente se comportaba como el robot real. Más adelante, se entrenó al agente con su nueva configuración, adaptando los entornos de aprendizaje cada vez más hasta acabar entrenando al agente en la resolución de laberintos.

Por último, se enfrentó a los modelos que aprendieron de los laberintos a un laberinto mayor con el fin de poder confirmar que el aprendizaje de los modelos era real, y que no se limitaba a resolver únicamente los entornos donde entrenaba.

Con todo ello, se destacan las siguientes conclusiones:

- Un robot como el NXT puede explorar un entorno laberíntico controlado por una red neuronal entrenada para ese propósito.
- El aprendizaje por refuerzo permite que el robot aprenda mediante la interacción con su entorno y sin necesidad de intervención humana.
- El aprendizaje adquirido en laberintos sencillos (3x3 celdas) se puede generalizar para resolver laberintos más complejos (5x5 celdas).
- A pesar de que el agente recibe una información limitada de su entorno y está controlado por una arquitectura sencilla, ha sido capaz de inferir comportamientos complejos propios de los seres humanos como la regla de la mano derecha para salir de laberintos.

Respecto a la interiorización de los conocimientos aprendidos al realizar este proyecto, queremos mencionar que hemos experimentado de primera mano la utilidad del aprendizaje por refuerzo aplicado

a la robótica, especificado en el área de la exploración de entornos.

Se han obtenido resultados positivos, pudiendo observar la trayectoria de aprendizaje de los modelos, destacando los puntos clave del entrenamiento. Se ha intentado comprender el comportamiento del modelo, condicionándolo con los sistemas de recompensa propios del aprendizaje por refuerzo, logrando rendimientos cada vez más positivos que han acabado resultando en una progresión del aprendizaje. Todo ello ha servido para comprender mejor el proceso de aprendizaje, desde los primeros pasos donde el agente no es capaz de solucionar el problema hasta una evolución donde el agente busca nuevas formas de mejorar sus propias marcas y lograr resoluciones más eficientes.

En conclusión, se han observado los puntos fuertes y débiles del aprendizaje por refuerzo, qué es capaz de lograr y cuáles son sus limitaciones, permitiéndonos desarrollar y adaptar entornos de aprendizaje con el objetivo de que el modelo consiga resolverlos de forma eficiente.

6.2. Trabajo futuro

Existen múltiples extensiones de este proyecto como pueden ser la aplicación de los algoritmos de aprendizaje por refuerzo en otro tipo de ámbitos distintos a la resolución de laberintos de estructura desconocida, que contengan una trayectoria de aprendizaje similar. Estos ámbitos pueden ser entornos donde se tenga que trazar una trayectoria detectando elementos clave, con ejemplos como la detección de objetivos específicos en terrenos utilizando satélites [48], o el entrenamiento de comportamientos colaborativos en enjambres de mini-robots [49].

Como se menciona al inicio del proyecto, este es uno de los dos Trabajos de Fin de Grado en esta línea de investigación, siendo el otro el encargado de integrar el robot real con un ordenador con el objetivo de poder implementar algoritmos complejos con él. Por lo tanto, se puede comprender que la evolución lógica de esta línea de investigación es la de unificar estos dos proyectos, implementando los modelos entrenados a través de simulaciones al robot real de LEGO® MINDSTORMS®. El objetivo a proponerse sería el de lograr resolver laberintos de estructura desconocida con el robot real, adaptando las entradas y salidas del modelo a las observaciones y acciones que puede generar el robot real. Por ello se estructuró de manera que permitiese implementar este tipo de adaptaciones de manera más sencilla (ver apartado 4.4).

La idea de poder entrenar un robot en un entorno virtual y aplicar el aprendizaje en la realidad es muy interesante, ya que el entrenamiento virtual es muy rápido y permite el entrenamiento en paralelo, en comparación con el entrenamiento con pruebas reales. Y es que este sería el propósito final de un proyecto mayor, que albergase tanto este Trabajo de Fin de Grado como la aplicación de los modelos de aprendizaje al robot real.

Bibliografía

[1] S. T. Leatherdale and J. Lee, "Artificial intelligence (ai) and cancer prevention: the potential application of ai in cancer control programming needs to be explored in population laboratories such as compass," *Cancer Causes & Control*, vol. 30, no. 7, pp. 671–675, 2019.

[2] H. Rodney, K. Valaskova, and P. Durana, "The artificial intelligence recruitment process: How technological advancements have reshaped job application and selection practices," *Psychosociological Issues in Human Resource Management*, vol. 7, no. 1, pp. 42–47, 2019.

[3] K. De Jong, "Learning with genetic algorithms: An overview," *Machine learning*, vol. 3, no. 2-3, pp. 121–138, 1988.

[4] A. Moreno, "Aprendizaje automático," 1994.

[5] A. L. Beam and I. S. Kohane, "Big data and machine learning in health care," *Jama*, vol. 319, no. 13, pp. 1317–1318, 2018.

[6] J. Brownlee, "14 different types of learning in machine learning," *Machine Learning Mastery*, vol. 11, no. 11, 2019.

[7] A. Dey, "Machine learning algorithms: a review," *International Journal of Computer Science and Information Technologies*, vol. 7, no. 3, pp. 1174–1179, 2016.

[8] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[9] M. Castelli, L. Vanneschi, and Á. Largo, "Supervised learning: classification," 2016.

[10] S. Xie and Y. Liu, "Improving supervised learning for meeting summarization using sampling and regression," *Computer Speech & Language*, vol. 24, no. 3, pp. 495–514, 2010.

[11] D. Pandey, K. Niwaria, and B. Chourasia, "Machine learning algorithms: A review," *Machine Learning*, vol. 6, no. 02, 2019.

[12] J. Brownlee, "Supervised and unsupervised machine learning algorithms," *Machine Learning Mastery*, vol. 16, no. 03, 2016.

[13] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017.

[14] K. J. Cios, R. W. Swiniarski, W. Pedrycz, and L. A. Kurgan, "Unsupervised learning: association rules," in *Data Mining*, pp. 289–306, Springer, 2007.

[15] X. J. Zhu, "Semi-supervised learning literature survey," 2005.

[16] J. R. Foulds and E. Frank, "A review of multi-instance learning assumptions," 2010.

[17] A. Zafra, S. Ventura, and E. Herrera-Viedma, "Aprendizaje multi-instancia con programación genética para web mining,"

[18] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," *arXiv preprint arXiv:1906.12340*, 2019.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[21] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[22] J. Denrell, C. Fang, and D. A. Levinthal, "From t-mazes to labyrinths: Learning from model-based feedback," *Management Science*, vol. 50, no. 10, pp. 1366–1378, 2004.

[23] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," *Robotics: Science and Systems, MIT Press Journal*, vol. 6, pp. 33–40, 2011.

[24] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[25] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[26] D. Garzón Ramos and M. Birattari, "Automatic design of collective behaviors for robots that can display and perceive colors," *Applied Sciences*, vol. 10, no. 13, p. 4654, 2020. Vídeos de demostración en http://iridia.ulb.ac.be/supp/IridiaSupp2019-008/.

[27] "Bricx command center." http://bricxcc.sourceforge.net/.

[28] J. J. Roldán, E. Peña-Tapia, P. Garcia-Aunon, J. Del Cerro, and A. Barrientos, "Bringing adaptive and immersive interfaces to real-world multi-robot scenarios: Application to surveillance and intervention in infrastructures," *Ieee Access*, vol. 7, pp. 86319–86335, 2019.

[29] J. J. Roldán, E. Peña-Tapia, D. Garzón-Ramos, J. de León, M. Garzón, J. del Cerro, and A. Barrientos, "Multi-robot systems, virtual reality and ros: developing a new generation of operator interfaces," in *Robot Operating System (ROS)*, pp. 29–64, Springer, 2019.

[30] V. H. Andaluz, F. A. Chicaiza, C. Gallardo, W. X. Quevedo, J. Varela, J. S. Sánchez, and O. Arteaga, "Unity3d-matlab simulator in real time for robotics applications," in *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pp. 246–263, Springer, 2016.

[31] "Unity 2019.3." https://unity.com/releases/2019-3.

[32] "Unity student plan, free 3d and vr software for students." https://store.unity.com/academic/unity-student.

[33] "Unity-technologies/ml-agents: Unity machine learning agents toolkit." https://unity.com/products/machine-learning-agents.

[34] "Machine learning bots for game development, reinforcement learning, unity." https://unity.com/products/machine-learning-agents.

[35] G. C. Lopes, M. Ferreira, A. da Silva Simões, and E. L. Colombini, "Intelligent control of a quadrotor with proximal policy optimization reinforcement learning," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in*

Education (WRE), pp. 503–508, IEEE, 2018.

[36] C.-C. Wong, S.-Y. Chien, H.-M. Feng, and H. Aoyama, "Motion planning for dual-arm robot based on soft actor-critic," *IEEE Access*, vol. 9, pp. 26871–26885, 2021.

[37] "Proximal policy optimization." https://openai.com/blog/openai-baselines-ppo/.

[38] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, Citeseer, 2000.

[42] "Soft actor critic, deep reinforcement learning with real-world robots." https://bair.berkeley.edu/blog/2018/12/14/sac/.

[43] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[44] "Repositorio con código de tfg, daniel cabañas, github." https://github.com/cabannas/TFG-Entrenamiento-Robot-Aprendizaje-Automatico.

[45] "Ml-agents getting started guide, unity, github." https://github.com/Unity-Technologies/ml-agents/tree/.

[46] "Ml-agents learning environment create new, unity, github." https://github.com/Unity-Technologies/ml-agents/blob/.

[47] "Videos de entrenamiento y resolución de laberintos, daniel cabañas, youtube." https://www.youtube.com/channel/UCByqlLGuc5LMO_yk07LmecQ/videos.

[48] B. Uzkent, C. Yeh, and S. Ermon, "Efficient object detection in large images using deep reinforcement learning," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1824–1833, 2020.

[49] P. Garcia-Aunon, J. J. Roldán, and A. Barrientos, "Monitoring traffic in future cities with aerial swarms: Developing and optimizing a behavior-based surveillance algorithm," *Cognitive Systems Research*, vol. 54, pp. 273–286, 2019.

Apéndices

A

Especificaciones del Modelo NXT del Robot de LEGO® MINDSTORMS®

A continuación, se incluye una tabla con las especificaciones del robot NXT de LEGO® MINDSTORMS® que se ha utilizado en este Trabajo de Fin de Grado.

Marca	LEGO®
Familia	MINDSTORMS®
Modelo	NXT
Versión	5ª
Generación	2ª
Fecha de Producción	2006
Procesador	Atmel AT91SAM7S256 (ARM7TDMI core)
Velocidad del Procesador	48 MHz
Pantalla	LCD monocromo 100x64 pixeles
Memoria Interna	256 KB Flash
Memoria RAM	64 KB
Bluetooth	Sí
WiFi	No
USB	Sí
Batería	Pila eléctrica AA 1,5V 2.500-3.000mAh (6)

Tabla A.1: Características del *brick* del robot NXT utilizado en este proyecto

B

Pseudocódigos de los Algoritmos PPO y SAC

B.1. PPO

Se presenta el pseudocódigo algoritmo de PPO, basado en el modelo *Actor-Critic*, según el artículo de OpenIA [40]:

```
for iteration = 1, 2, ... do
    for actor = 1, 2, ... to N do
        Run policy π_θold in environment for T timestamps
        Compute advantage estimates A_1, ..., A_T
    end
    Optimize surrogate L wrt θ, with K epochs and minibatch size M <= NT
    θ_old ← θ
end
```

Algoritmo B.1: Pseudocódigo del algortimo de PPO utilizado en este proyecto

B.2. SAC

A continuación, se presenta el pseudocódigo del algoritmo de SAC según el estudio de Haarnoja et al. [43]:

input : θ_1, θ_2, ϕ
$\overline{\theta}_1 \leftarrow \theta_1, \overline{\theta}_2 \leftarrow \theta_2$
$D \leftarrow \emptyset$
for *each iteration* **do**
  for *each environment step* **do**
    $a_t \sim \pi_\phi(a_t \mid s_t)$
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
    $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end
  for *each gradient step* **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
    $\alpha \leftarrow \alpha - \lambda \nabla_\alpha J(\alpha)$
    $\overline{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\overline{\theta}_i$ for $i \in \{1, 2\}$
  end
end
output: θ_1, θ_2, ϕ

Algoritmo B.2: Pseudocódigo del algortimo de SAC utilizado en este proyecto

C

Ejemplo de Procedimiento en Bricx Command Center

A continuación, se muestra un fragmento de un procedimiento del robot NXT usando el entorno de desarrollo Bricx Command Center.

Código C.1: Código de ejemplo de un programa NXC de Bricx Command Center

```
1  task go_forward()
2  {
3    for (int i = 0; i < 2; i++)
4    {
5        OnFwd(OUT_A, 100);
6        OnFwd(OUT_B, 100);
7
8        Wait(1000);
9
10       OnFwd(OUT_A, 0);
11       OnFwd(OUT_B, 0);
12
13       Wait(1000);
14   }
15 }
16
17 task go_backward()
18 {
19   for (int i = 0; i < 2; i++)
20   {
21       OnFwd(OUT_A, -100);
22       OnFwd(OUT_B, -100);
23
24       Wait(1000);
25
26       OnFwd(OUT_A, 0);
27       OnFwd(OUT_B, 0);
28
29       Wait(1000);
30   }
31 }
```

D

Tabla de Mediciones del Robot Real

A continuación, se adjuntan las tablas con las mediciones realizadas para establecer las velocidades que alcanza el robot NXT en función de la potencia de los servomotores izquierdo y derecho. La primera super-tabla (ver cuadro D.1 y cuadro D.2) contiene las mediciones realizadas con la misma potencia establecida en ambos servomotores, mientras que el cuadro D.3 contiene las mediciones realizadas con el motor izquierdo a 100 de potencia y variaciones en el motor derecho, y el cuadro D.4 contiene las mediciones realizadas con el motor derecho a 100 de potencia y variaciones en el motor izquierdo.

El tiempo de cada iteración tiene una duración de 1000 ms, las distancias se miden en cm y los ángulos en grados.

Iteración	100	90	80	70	60	50	40	30	20	10
1	25	21,9	18,8	16,2	14	11	9,2	6,7	3,6	0,9
2	24,5	22	19,5	16,5	13,6	11,1	9,1	5,6	3,8	1
3	25,5	21,1	19	16	13,7	10,7	8,7	5,5	3,6	0,2
4	25	21,2	18,5	15,8	13,4	10,8	8,5	5,5	3,4	0
5	24,8	21,3	19,1	16	13,2	11	8,8	5,5	3,3	0
6	25,7	22	18,9	16,2	13,3	11,1	8,7	6,2	3,4	0,5
7	24,8	21,8	18,7	16,1	13,4	11,1	8,4	5,8	3,5	0,9
8	25	21	18,7	16,2	13,4	10,9	8,6	5,5	3,3	0,5
9	25	21,4	18,8	15,8	13,6	11	9,1	6,3	3,8	0,1
10	24,6	21,6	19	16	13,4	11,1	8,6	5	3,5	0
11	24,4	21,8	18,7	16	13,4	11,2	9,6	6,1	3,3	0
12	24,3	21,4	18,5	15,8	13,6	11	9	5,9	3,4	0,2

Tabla D.1: Medición de distancia recorrida con la misma variación en la potencia de ambos servomotores (parte 1)

13	24,5	22	18,8	15,9	13,5	10,9	8,6	5,4	3,6	0
14	24,7	21,6	19,2	16,1	13,5	10,7	8,5	5,6	3,6	0,5
15	25	21,2	18,5	16,1	13,5	11,1	8,8	5,9	3,5	0
16	24,6	21,3	18,7	15,9	13,4	11,2	8,7	6	3,5	0
17	24,8	21,5	18,5	16	13,7	11	8,4	5,8	3,4	0,2
18	25	21,7	18,3	16,1	13,4	11,1	8,5	5,7	3,4	0
19	25,5	21,5	18,4	16,3	13,5	10,9	8,7	5,5	3,5	0
20	24,5	20,9	18,5	16,4	13,4	10,8	8,5	5,5	3,4	0
21	24,6	21,3	19	15,5	13,2	10,9	8,6	5,5	3,3	0,2
22	25	21,7	19	15,8	13,5	10,8	8,6	5,3	3,4	0,2
23	25,5	21,8	18,9	16	13,2	10,9	8,4	5,8	3,4	0
24	24	22	19,1	15,9	13,4	11,1	8,5	5,7	3,6	0
25	24,8	21,4	18,6	15,7	13,7	10,8	8,7	5,8	3,5	0
26	24,5	21,3	18,4	15,8	13,6	10,8	8,6	6	3,3	0,5
27	24,3	21,2	18,5	15,6	13,1	10,9	8,4	5,8	3,5	0,3
28	24,1	21,4	18,3	15,5	13,4	11	8,3	5,8	3,8	0
29	24,2	21,4	19	16,2	13,3	10,8	9	5,6	3,6	0
30	25	21,1	18,8	16	13,3	11	8,8	5,8	3,7	0
31	24,2	21,2	18,7	15,6	13,5	10,9	8,8	5,8	3,4	0,1
32	24,3	21,3	18,8	15,4	13,4	11	8,7	5,8	3,3	0
33	25	21,6	18,4	15,7	13,1	11,1	8,4	6	3,6	0
34	24,4	21	18,4	15,9	13,2	11,1	8,6	5,9	3,6	0,2
35	25,3	20,5	18,8	15,9	13,6	11	8,7	5,8	3,3	0
36	24,1	21,5	18,7	15,7	13,4	11,1	8,7	5,9	3,4	0
37	24,6	20,7	18,5	15,7	13,4	10,9	8,7	5,3	3,4	0,2
38	24,4	21,6	18,5	15,8	13,5	11,2	8,4	6,1	3,5	0
39	24,8	21	18,7	15,4	13,4	11,2	8,4	6	3,7	0,3
40	24,2	21,4	19	15,6	13,2	11	8,6	5,9	3,8	0,1
Media:	24,71	21,41	18,73	15,90	13,43	10,98	8,67	5,76	3,49	0,17
Diferencia:		3,29	2,68	2,82	2,47	2,45	2,30	2,90	2,26	3,32

Tabla D.2: Medición de distancia recorrida con la misma variación en la potencia de ambos servomotores (parte 2)

Iteración	90	80	70	60	50	40	30	20	10	0
1	20,5	20	17	15	12,5	11	9	8	6	5
2	21	19	17	16	12	11,5	9	8	6	5,5
3	20	19,5	16,8	15	12,5	11	9,5	7,5	6,2	5,5
4	20,2	19	16,5	15	12	11,5	8,7	7,5	6	5
5	20,5	18,5	17	15,5	12,5	10,8	8,9	7,7	6,2	5,2
6	21	20	17,5	15	12	10,9	9	8	5,8	5,5
7	20	19	17	15,2	12,2	11	8,7	7,8	6	5,5
8	20	19	18	15	12,8	11,2	9	7,5	6	5,3
9	21	18,5	17,7	15,7	12	10,8	9,3	7,7	6,2	5
10	20,7	19,7	17	15	12	11	9	7,5	6	5,2
Media	20,49	19,22	17,15	15,24	12,3	11,1	9,01	7,72	6,04	5,27
grados/s	6,43	9,47	14,52	20	27	33,3	40	46,5	52,5	58,3

Tabla D.3: Medición de distancia recorrida con variaciones en la potencia del servomotor derecho

Iteración	90	80	70	60	50	40	30	20	10	0
1	22	21	18,5	17	14	12	9	7,5	4,5	4
2	21,5	21	18,5	17	14,5	10,5	9	8	5	4,5
3	21	21,5	19	16,5	14,5	11,5	9,5	7,5	5	3,5
4	22,5	21	19	16	13	11,5	9	7,5	5,5	4
5	22	21,2	18,5	16	14	11	9	7	5	4,5
6	22,5	21	18,5	16	15	11	9	7	5,5	4
7	21,5	21,5	19	17	14	11	9	7,5	5	4,2
8	21	21	19	17	13,5	12	8,5	7	5	5
9	22	21,1	18,5	16	14	11,5	9	7,5	5	5
10	21,5	21	19	16	14	11,5	9,5	6,5	5,5	4
Media	21,75	21,13	18,75	16,45	14,1	11,4	9,05	7,3	5,1	4,27
grados/s	4,39	9	12,86	18	24,4	32,8	36,2	42,5	50	55

Tabla D.4: Medición de distancia recorrida con variaciones en la potencia del servomotor izquierdo

E

Tabla de Recompensas de los Modelos Entrenados

A continuación, se muestra la tabla de recompensas utilizada en los modelos entrenados del proyecto.

	Acción	Valor del Refuerzo
1	Cualquier movimiento menos retroceder	-0.00001f
2	Retroceder	-0.0001
3	Alcanzar un objetivo	+(targetsReached/totalTargets)
4	Alcanzar todos los objetivos	+1
5	Encontrar una pared cercana y no retroceder	-0.0002
6	Encontrar una pared cercana y retroceder	+0.0001
7	Chocar con una pared	-0.001

Tabla E.1: Valores de los refuerzos aplicados al modelo de aprendizaje

F

Código de Generación de Laberintos Dinámicos

A continuación, se muestran fragmentos del código utilizado para la generación de laberintos pseudo-aleatorios de dimensiones 2x2 y 3x3.

En dimensiones de 2x2 la única manera de no cercar un recorrido es implementando una única pared a la vez de las cuatro existentes, por ello simplemente se elige una utilizando un *random* y el resto se deshabilitan.

En dimensiones de 3x3 se eligen primero 2 paredes centrales, y en función de cuáles se hayan elegido eligiremos las demás, evitando delimitar el recorrido total del laberinto.

Código F.1: Código de generación de los laberintos de dimensión 2x2

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Random = UnityEngine.Random;


public class Wall_Control_2 : MonoBehaviour
{
    public GameObject obj_outer_wall_1;
    public GameObject obj_outer_wall_2;
    public GameObject obj_outer_wall_3;
    public GameObject obj_outer_wall_4;

    private List<GameObject> outer_walls_list = new List<GameObject>();

    void Start()
    {
      outer_walls_list.Add(obj_outer_wall_1);
      outer_walls_list.Add(obj_outer_wall_2);
      outer_walls_list.Add(obj_outer_wall_3);
      outer_walls_list.Add(obj_outer_wall_4);
    }

    void Update()
    {
      foreach (GameObject wall in outer_walls_list){
          wall.SetActive(false);
      }
      int randWall;

      // Solo se puede activar un muro
      randWall = Random.Range(0,4);

      outer_walls_list[randWall].SetActive(true);
    }
}
```

Código F.2: Código de generación de los laberintos de dimensión 3x3 (parte 1)

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Random = UnityEngine.Random;
5
6  public class Wall_Control_3 : MonoBehaviour
7  {
8      public GameObject obj_inner_wall_1;
9      public GameObject obj_inner_wall_2;
10     public GameObject obj_inner_wall_3;
11     public GameObject obj_inner_wall_4;
12
13     public GameObject obj_outer_wall_1;
14     public GameObject obj_outer_wall_2;
15     public GameObject obj_outer_wall_3;
16     public GameObject obj_outer_wall_4;
17     public GameObject obj_outer_wall_5;
18     public GameObject obj_outer_wall_6;
19     public GameObject obj_outer_wall_7;
20     public GameObject obj_outer_wall_8;
21
22     private List<GameObject> inner_walls_list = new List<GameObject>();
23     private List<GameObject> outer_walls_list = new List<GameObject>();
24
25     void Start()
26     {
27       inner_walls_list.Add(obj_inner_wall_1);
28       inner_walls_list.Add(obj_inner_wall_2);
29       inner_walls_list.Add(obj_inner_wall_3);
30       inner_walls_list.Add(obj_inner_wall_4);
31
32       outer_walls_list.Add(obj_outer_wall_1);
33       outer_walls_list.Add(obj_outer_wall_2);
34       outer_walls_list.Add(obj_outer_wall_3);
35       outer_walls_list.Add(obj_outer_wall_4);
36       outer_walls_list.Add(obj_outer_wall_5);
37       outer_walls_list.Add(obj_outer_wall_6);
38       outer_walls_list.Add(obj_outer_wall_7);
39       outer_walls_list.Add(obj_outer_wall_8);
40     }
41
42     void Update()
43     {
44       // Primero los 4 muros internos activados y los 8 externos desactivados
45       foreach (GameObject wall in inner_walls_list){
46           wall.SetActive(true);
47       }
48       foreach (GameObject wall in outer_walls_list){
49           wall.SetActive(false);
50       }
```

Código F.3: Código de generación de los laberintos de dimensión 3x3 (parte 2)

```
51        // Elegimos 2 padres al azar y los abrimos
52        int randInner1, randInner2;
53        randInner1 = Random.Range(0,4);
54
55        do{
56          randInner2 = Random.Range(0,4);
57        }while (randInner1 == randInner2);
58
59        inner_walls_list[randInner1].SetActive(false);
60        inner_walls_list[randInner2].SetActive(false);
61
62        int randOuter1, randOuter2;
63        randOuter1 = Random.Range(0,4);
64        randOuter2 = 0; // Necesario definirlo para Unity
65
66        // A1 y B1 son los hijos de randInner1
67        // A2 y B2 son los hijos de randInner2
68        // A1 = 0; B1 = 1; A2 = 2; B2 = 3;
69
70        // A continuación se establecen una serie de restricciones entre los muros
71        // generados aleatoriamente para evitar cerrar celdas
72        if(((randInner1 == 0 || randInner1 == 2) &&
73        (randInner2 == 0 || randInner2 == 2))
74        ||
75        ((randInner1 == 1 || randInner1 == 3) &&
76        (randInner2 == 1 || randInner2 == 3))){
77
78        //Si salen las opciones 1-3 o 2-4
79          if (randOuter1 == 0){ // No puede ser A1-A1 o A1-B2
80            do{
81              randOuter2 = Random.Range(0,4);
82            }while ((randOuter1 == randOuter2) || (randOuter2 == 3));
83          }
84          if (randOuter1 == 1){ // No puede ser B1-B1 o B1-A2
85            do{
86              randOuter2 = Random.Range(0,4);
87            }while ((randOuter1 == randOuter2) || (randOuter2 == 2));
88          }
89          if (randOuter1 == 2){ // No puede ser A2-A2 o A2-B1
90            do{
91              randOuter2 = Random.Range(0,4);
92            }while ((randOuter1 == randOuter2) || (randOuter2 == 1));
93          }
94          if (randOuter1 == 3){ // No puede ser B2-B2 o B2-A1
95            do{
96              randOuter2 = Random.Range(0,4);
97            }while ((randOuter1 == randOuter2) || (randOuter2 == 0));
98          }
99
100       }else{ // Si salen las opciones 1-2, 1-4, 2-3 y 3-4
```

Código F.4: Código de generación de los laberintos de dimensión 3x3 (parte 3)

```
101             if (randOuter1 == 0){ // No puede ser A1-A1 o A1-A2
102               do{
103                 randOuter2 = Random.Range(0,4);
104               }while ((randOuter1 == randOuter2) || (randOuter2 == 2));
105             }
106             if (randOuter1 == 1){ // No puede ser B1-B1 o B1-B2
107               do{
108                 randOuter2 = Random.Range(0,4);
109               }while ((randOuter1 == randOuter2) || (randOuter2 == 3));
110             }
111             if (randOuter1 == 2){ // No puede ser A2-A2 o A2-A1
112               do{
113                 randOuter2 = Random.Range(0,4);
114               }while ((randOuter1 == randOuter2) || (randOuter2 == 0));
115             }
116             if (randOuter1 == 3){ // No puede ser B2-B2 o B2-B1
117               do{
118                 randOuter2 = Random.Range(0,4);
119               }while ((randOuter1 == randOuter2) || (randOuter2 == 1));
120             }
121           }
122
123           int padre1 = randInner1 + 1;
124           int padre2 = randInner2 + 1;
125           int hijo1, hijo2;
126
127           if (randOuter1 == 0){
128             hijo1 = 2 *padre1 -1;
129           }else if (randOuter1 == 1){
130             hijo1 = 2 *padre1;
131           }else if (randOuter1 == 2){
132             hijo1 = 2 *padre2 -1;
133           }else{
134             hijo1 = 2 *padre2;
135           }
136
137           if (randOuter2 == 0){
138             hijo2 = 2 *padre1 -1;
139           }else if (randOuter2 == 1){
140             hijo2 = 2 *padre1;
141           }else if (randOuter2 == 2){
142             hijo2 = 2 *padre2 -1;
143           }else{
144             hijo2 = 2 *padre2;
145           }
146
147           outer_walls_list[hijo1 -1].SetActive(true);
148           outer_walls_list[hijo2 -1].SetActive(true);
149         }
150     }
```

G

Prueba 1: Hiperparámetros

A continuación, se adjuntan los hiperparámetros utilizados en las pruebas de configuración de movimiento.

Código G.1: Hiperparámetros de la Prueba 1 para la configuración de movimiento

```
Prueba1:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 32
    buffer_size: 2048
    learning_rate_schedule: linear
    hidden_units: 128
    num_layers: 2
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

H

Prueba 2: Hiperparámetros

A continuación, se adjuntan los hiperparámetros utilizados en las pruebas de configuración de sensores.

Código H.1: Hiperparámetros de la Prueba 2 para la configuración de sensores

```
Prueba2:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 32
    buffer_size: 2048
    learning_rate_schedule: linear
    hidden_units: 128
    num_layers: 2
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

Prueba 3: Hiperparámetros

I.1. Prueba 3.1

A continuación, se adjuntan los hiperparámetros utilizados en las pruebas con laberintos de dimensiones 2x2.

I.1.1. Configuración sin el Módulo de Curiosidad

Código I.1: Hiperparámetros de la Prueba 3.1 con laberintos de dimensiones 2x2 sin el módulo de curiosidad

```
Prueba2:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 32
    buffer_size: 2048
    learning_rate_schedule: linear
    hidden_units: 128
    num_layers: 2
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

I.1.2. Configuración con el Módulo de Curiosidad

Código I.2: Hiperparámetros de la Prueba 3.1 con laberintos de dimensiones 2x2 con el módulo de curiosidad

```
Prueba3_1_curiosity:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 1024
    buffer_size: 10240
    learning_rate_schedule: linear
    hidden_units: 128
    num_layers: 2
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
        curiosity:
            strength: 0.1
            gamma: 0.99
            encoding_size: 256
```

I.2. Prueba 3.2

A continuación, adjuntamos los hiperparámetros utilizados en las pruebas con laberintos de dimensiones 3x3.

I.2.1. Configuraciones con y sin el Módulo de Curiosidad

Código I.3: Hiperparámetros de la Prueba 3.2 con laberintos de dimensiones 3x3 sin el módulo de curiosidad y con redes que contienen 3 capas ocultas con 256 unidades por capa

```
Prueba3_2_no_curiosity:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 1024
    buffer_size: 10240
    learning_rate_schedule: linear
    hidden_units: 256
    num_layers: 3
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
```

Código I.4: Hiperparámetros de la Prueba 3.2 con laberintos de dimensiones 3x3 con el módulo de curiosidad y con redes que contienen 3 capas ocultas con 256 unidades por capa

```
Prueba3_2_curiosity:
    summary_freq: 50000
    time_horizon: 64
    max_steps: 4e7
    learning_rate: 3.0e-4
    batch_size: 1024
    buffer_size: 10240
    learning_rate_schedule: linear
    hidden_units: 256
    num_layers: 3
    normalize: false
    beta: 5.0e-3
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    reward_signals:
        extrinsic:
            gamma: 0.99
            strength: 1.0
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID